UNIVERSIDADE DO VALE DO TAQUARI

CURSO DE ENGENHARIA DA COMPUTAÇÃO

APLICAÇÃO DE *FIREWALL* COM FILTRO AVANÇADO PARA APLICAÇÕES WEB

Ismael Branco

Lajeado, novembro de 2020

Ismael Branco

APLICAÇÃO DE *FIREWALL* COM FILTRO AVANÇADO PARA APLICAÇÕES WEB

Monografia apresentada na disciplina de Trabalho de Conclusão de Curso, do curso de Engenharia da Computação, da Universidade do Vale do Taquari – Univates, como parte da exigência para a obtenção do título de Bacharel.

Orientador: Edson Moacir Ahlert

Lajeado, novembro de 2020

RESUMO

O presente trabalho de conclusão de curso objetiva o estudo e implementação de uma aplicação de filtro avançado de pacotes de dados na porta 80 e 443 para aplicações *web,* gerando segurança nas aplicações. O estudo da ferramenta é referenciado ao longo do trabalho com demandas de leis, seguranças, normas e possíveis tipos de ataques a aplicação *web*. A aplicação utilizada para os testes de vulnerabilidades é da empresa AWTI Sistemas de Informação a qual não possui proteção para quaisquer vulnerabilidades de ataques ao sistema *web*. A prevenção dos ataques ao servidor Apache se dá pelo módulo adicional ModSecurity em conjunto com o módulo Mod Evasive. Para interromper os ataques o ModSecurity conta com regras de vulnerabilidades pré-determinadas, enquanto para evitar novos ataques do mesmo endereço de IP o Mod Evasive bloqueia por tempo determinado o acesso deste atacante. Comparando os resultados de pré-implementação e pós-implementação destes dois módulos do Apache, as vulnerabilidades da aplicação web testada nas portas 80 e 443 foram minimizadas em todas as vulnerabilidades encontradas anteriormente, defendendo os dados pessoais dos clientes e impactando também na melhora de performance da aplicação, em caso de ataque.

Palavras-chave: ModSecurity. Mod Evasive. Vulnerabilidades.

ABSTRACT

The present work of conclusion of the course aims at the study and implementation of an application of advanced filter of data packets on port 80 and 443 for web applications, generating security in the applications. The study of is referenced throughout the work with demands for laws, security tools, standards and types of web application options. The application used for the vulnerability tests is from the company AWTI Sistemas de Informação which has no protection for any web system vulnerabilities. Apache servers are prevented by the additional ModSecurity module in conjunction with the Mod Evasive module. To stop ModSecurity attacks it has predetermined vulnerability rules, while to prevent new attacks from the same IP address or Mod Evasive blocks this attacker for a specific time or access. Comparing the results of pre-implementation and post-implementation of these two Apache modules, how vulnerabilities of the web application tested on ports 80 and 443 were minimized in all the vulnerabilities found previously, defending the personal data of customers and also impacting on the performance improvement application in case of attack.

Keywords: ModSecurity. Mod Evasive. Vulnerabilities.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Desde o surgimento da internet, o desenvolvimento de sites e aplicativos para *web* vem crescendo rapidamente, juntamente com desenvolvimento das redes de computadores e internet, possibilitando o acesso destes sites e aplicativos da *web* de qualquer local do mundo. Segundo estatísticas, o desenvolvimento de soluções para *web* vem crescendo de forma exponencial (SILVA, 2020), podendo ter aumentado ainda mais nos últimos anos, de forma que mesmo serviços bancários, de compras e vendas de produtos e serviços governamentais, agora estão disponíveis via *web*.

Aplicativos *web* vem crescendo rapidamente devido a praticidade de disponibilizar informações, serviços governamentais, serviços bancários, negociação e comércio de produtos e serviços dos mais diversos.

Aplicativos *web* atendem a requisitos de programação com a praticidade de poder acessar de qualquer lugar e ainda serem multiplataforma, juntamente com a evolução das redes mundiais e serviços locais de infraestrutura de rede, o que vem ocorrendo devido a demandas de uso.

Uma aplicação de computador na empresa ou comércio, gerencia os dados do negócio de forma organizada, facilitando e agilizando o trabalho de forma que seja viável possuí-la. Porém estas aplicações utilizam dados pessoais que podem ficar vulneráveis ao estarem em computadores com internet, ainda mais vulneráveis se o sistema for aberto para acessos externos à rede local (MONTEVERDE; CAMPIOLO, 2014).

Em meados de 2000, quando a internet já ganhava um grande aumento na utilização com demandas de aplicativos *web*, havia a disposição de muitas

plataformas de sistemas operacionais, com diferenças de programação e compatibilidades de aplicações, ou seja, muitas vezes a plataforma de cada sistema operacional é incompatível com a do outro, assim impossibilitando o compartilhamento de aplicativos entre eles, tendo de criar um novo aplicativo na linguagem específica do sistema operacional (AUGUSTO, 2017).

Desde então surgiram novas linguagens de programação, muitas delas multiplataformas, possibilitando a utilização entre diversos equipamentos com sistemas operacionais diferentes. Entre elas temos a multiplataforma *web*, programação para internet ou intranet, desta forma há possibilidade de acessar seu aplicativo desde qualquer sistema operacional.

Não existe um órgão público ou privado que demonstre o fator de crescimento de cada linguagem para uma comparação do crescimento de multiplataformas, pois muitos dos aplicativos não são registrados em algum órgão de controle, mas embasado na popularidade e necessidade do comércio. A multiplataforma *web* vem sendo escolhida pela maioria das empresas de desenvolvimento, por se tratar de uma plataforma que não necessita instalação no dispositivo do cliente, assim possibilitando uma migração e acesso à aplicação muito fácil e rápida.

Todo aplicativo do mais simples ao mais complexo, armazena dados de clientes, muitas vezes dados importantes, que não podem ser disponibilizados ao público. Para isso existem normas de segurança, boas práticas e leis de proteção aos dados do cliente. Leis às quais empresas de desenvolvimento de *software* precisam observar, inclusive algumas delas devem ser seguidas mesmo por terceiros, para eventualidades, como por exemplo se caso acontecer algum ataque aos dados, perda de informação, ou vazamento de dados privados.

No Brasil, temos algumas leis e normas que devem ser seguidas na hora da criação de uma aplicação, tanto que, em 18 de Setembro de 2020, entrou em vigor mais uma lei brasileira, a qual representa a lei geral de proteção de dados, conhecida como LGPD (LEI Nº 13.709/18).

Com leis cada vez mais rigorosas com a proteção de dados de terceiros nas aplicações, e pelo crescimento constante de insegurança e ataques aos aplicativos de todas as plataformas, com potencial à plataforma *web*, vem crescendo a

importância da segurança e surgindo novas tecnologias para assegurar os dados de todos aplicativos *web*, visto que, em novembro de 2019, quase 80% das residências brasileiras já possuíam conexão com a internet, ou seja, 166 milhões de brasileiros têm acesso a internet (IBGE, 2019).

Segundo dados estatísticos apresentados pelo Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil, em 2019 os casos de ataque à páginas *web*, foram mais de 22 mil ataques apenas no protocolo HTTP, isso em servidores brasileiros (CERT.BR, 2019). Estes ataques se mantêm frequentes, juntamente com outros ataques indiretamente às páginas *web*, como em 2017, a NETSCOUT (2018) registrou um total de 264.900 ataques DDoS (ataques de negação de serviços) apenas no Brasil.

O fator de insegurança em grandes empresas e sistemas, como mídias sociais, sistemas bancários, sistemas governamentais, entre outros, vem sendo preocupante e de alto investimentos das empresas, para minimizar estes ataques, e assegurar os dados dos sistemas.

A segurança dos dados, na maioria das empresas de pequeno ou médio porte, é feita por um *firewall* baseado em *software* ou *hardware*, estes por sua vez contêm assistentes de instalação com políticas de segurança predefinidas, filtragem do tráfego IP recebido e enviado, bloqueios automáticos de compartilhamentos de arquivos, invisibilidade de portas TCP/IP, alertas e registros e entre outras funcionalidades (FORD, 2002). Porém, estes *firewalls* não implementam a segurança adequada para aplicações *web* (OWASP, 2017).

Para maior segurança de aplicações *web*, o indicado é o uso da ferramenta de *Web Advanced Filter* (*WAF*), ferramenta definida como um *hardware* ou *software* que protege uma ou mais aplicações *web* de ataques que exploram vulnerabilidades de aplicações *web*. Diferentemente de um *firewall* tradicional o sistema *WAF* protege as aplicações de ataques que exploram, especificamente, as vulnerabilidades de aplicações *web*. Sendo sua função inspecionar pacotes de dados transmitidos entre usuários e a aplicação *web*, baseado em regras predefinidas pelo operador da aplicação *WAF*. Ressalta-se que a aplicação *WAF*, é uma maneira de minimizar os impactos dos ataques, e não uma ferramenta que elimina todas as vulnerabilidades de uma aplicação *web* (IPA, 2011).

O presente trabalho visa minimizar as vulnerabilidades de qualquer aplicação *web*, com a implementação e avaliação de um *WAF*, com o objetivo de atender as normas e leis de segurança para um sistema de pequena ou média empresa, sem altos custos e acessível ao empreendedor. O *WAF* será implementado em uma aplicação multiplataforma, com linguagem PHP e base de dados de testes, para não comprometer dados reais de terceiros. O *WAF* será avaliado por *softwares* reconhecidos por testes de vulnerabilidades.

1.1 Tema e problema

O tema foi escolhido e embasado na nova lei geral de proteção de dados (LGPD), que entrou em vigor em setembro de 2020, para minimizar problemas de segurança em aplicações *web* de uma empresa de desenvolvimento de *software, a* qual atualmente não demanda nenhuma segurança à suas aplicações, mas deseja se adequar às normas, não só atendê-las, como também manter o sigilo e segurança dos dados de terceiros presentes em suas aplicações.

A ferramenta de aplicação *WAF* utilizada, poderá ser implementada em quaisquer pequenas e médias empresas com aplicações *web*, empresas as quais não costumam investir dinheiro na segurança de sua rede, mas necessitam a multiplataforma *web* para acessos externos de seu sistema. Com a aplicação do *WAF*, os ataques a essas empresas podem ser minimizados e manter baixo o custo de implementação, o que traz relevância para a escolha do tema, de forma que atenda o problema e possa ser utilizado em outros clientes, sem a necessidade de muitas alterações em sua configuração.

1.2 Objetivos

O objetivo deste trabalho é estudar e implementar um *WAF - Web Advanced Filter*, como forma de minimizar problemas de segurança da aplicação *web* desenvolvida pela empresa AWTI Sistemas de Informação. Para isso, criar uma

máquina virtual ou física com a aplicação de um *WAF* é a solução, com regras predefinidas em sua *blacklist*, minimizando as vulnerabilidades de aplicativos *web*, de forma que a solução poderá ser utilizada em qualquer sistema e não necessariamente ficando presa ao sistema da empresa de desenvolvimento citada neste trabalho.

1.3 Objetivos específicos

Comparar os resultados da implementação do *WAF* em uma máquina virtual, juntamente com um *software web*, para averiguar se as vulnerabilidades de segurança são minimizadas.

1.4 Estrutura do trabalho

O trabalho foi dividido em seis capítulos, o primeiro deles a referida introdução, o capítulo dois com referência teóricas, demandando uma breve revisão de conceitos de aplicações *web*, conteúdos sobre vulnerabilidades e normas de segurança, necessários para dar ênfase à compreensão de implementação de um *WAF*.

Ainda neste capítulo são tratadas as vulnerabilidades existentes em um sistema *web* e os ataques que vêm acontecendo, com finalidade de embasar o assunto, onde será apresentada a ferramenta de *ModSecurity* para filtragem de aplicações *web* e suas funcionalidades, respectivamente os conceitos, as ferramentas necessárias, e por fim será descrito o funcionamento de um *WAF*.

No terceiro capítulo, trabalhos com assuntos relacionados em *WAF*. O quarto capítulo, a metodologia de pesquisa utilizada, seguida pelo capítulo cinco, de resultados e discussão, e, para terminar, o capítulo seis, com as considerações finais, seguida da finalização do trabalho, com o capítulo de referências.

2 REVISÃO TEÓRICA

Para o entendimento da necessidade da aplicação do *WAF* e sua configuração, torna-se necessário o embasamento de alguns conteúdos para a compreensão do serviço e sua implementação. Para isso, serão abordados nos capítulos a seguir alguns conteúdos de base, visando elucidar a compreensão do assunto.

2.1 Segurança das informações de aplicativos *web*

É possível definir a segurança da informação em cinco partes: confidencialidade, integridade, disponibilidade, irretratabilidade e autenticidade. A disponibilidade é assegurar acesso em um uso rápido e confiável da informação, sempre que solicitado pela pessoa com autorização (NOLETO, 2020).

Para Noleto (2020), a confidencialidade objetiva preservar restrições autorizadas sobre acesso e divulgação de informações, onde o acesso aos dados deve ser gerido pela pessoa autorizada, assim como a visualização deles também deve ser permitida apenas para as pessoas autorizadas.

Ainda segundo Noleto (2020), a integridade dos dados, por sua vez, consiste em prevenir contra a modificação ou destruição imprópria de informação, ou seja, apenas as pessoas autorizadas podem alterar os dados.

Segundo Noleto (2020), a irretratabilidade é a responsabilidade de um terceiro possuir os dados de alguém, se responsabilizando pelos dados ou documentos.

Noleto (2020) ainda define autenticidade como todo arquivo e documento

existente em uma empresa tenha registrado a autoria de origem e as últimas modificações de cada documento ou dado.

Toda aplicação nasce com vulnerabilidades de segurança, um sistema *web*, por exemplo, se não tratadas as brechas de segurança como *Buffer Overflow, SQL Injection, Credential Stuffing* e *Session Hijacking*, se torna extremamente vulnerável e deixa de ser uma aplicação confiável (MONTEIRO, 2015).

A vulnerabilidade de *Buffer Overflow* ou transbordamento de dados é quando acontece um estouro no armazenamento de dados de um programa, e ele é obrigado a utilizar memória adjacente, deixando esses dados em memória adjacente vulneráveis (STALLINGS; BROWN, 2015).

A vulnerabilidade de *SQL Injection* ou inserção de código de banco de dados é quando acontece uma inserção de código em linguagem de banco de dados em um campo de digitação do usuário, e não há tratamento de código, deixando executar este comando diretamente no banco de dados (STALLINGS; BROWN, 2015).

A vulnerabilidade de *Credential Stuffing* é quando o atacante utiliza a senha e usuário de um cliente em um serviço A para atacar com mesmo usuário e senha no serviço B (ROHR, 2019).

A vulnerabilidade de *Session Hijacking* ou roubo de sessão de usuário é quando o atacante ataca o navegador do cliente e rouba a sessão de usuário e possivelmente senha logada para acessar o sistema (LOUIS, 2011).

2.1.1 *OWASP* (*Open Web Application Security*)

A instituição *OWASP* é uma fundação de estudo, a qual não visa fins lucrativos, que se empenha em ensinar e avaliar empresas e organizações de programação para desenvolver aplicações confiáveis. O foco maior da *OWASP* são aplicações *web*.

Seu método de avaliação da segurança é determinado por reunir informações que permitam avaliar os riscos de segurança das informações e assim, combater os ataques de segurança das aplicações. Todo seu estudo das informações e seus métodos de segurança são disponibilizados, gratuitamente, na *internet*, para qualquer organização. Seus estudos são utilizados como referência em organizações reconhecidas, como *U.S. Defense Information Systems Agency* (DISA) e *U.S. Federal*

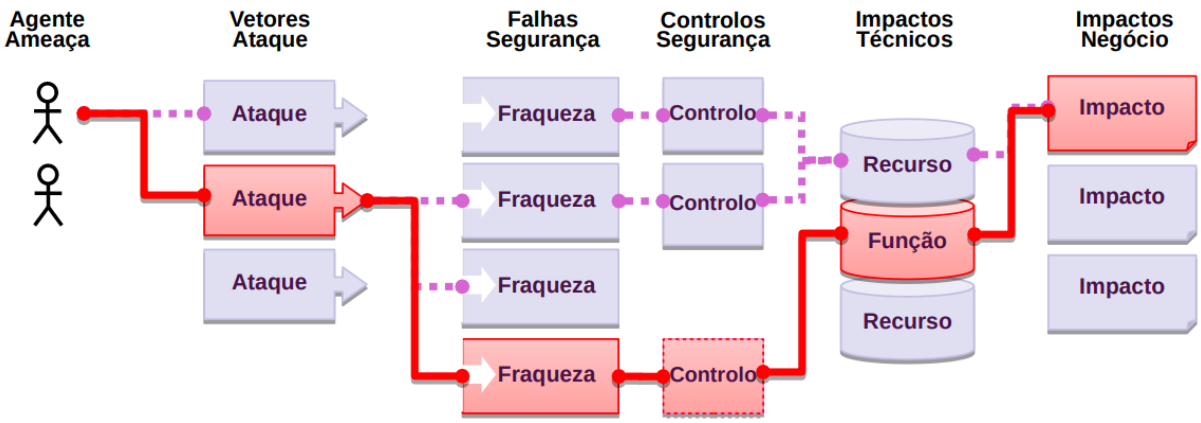
Trade Commissione PCI Council (MONTEVERDE; CAMPIOLO, 2014).

OWASP leva em conta, em seus estudos, avaliações locais de cada país, juntamente com normas de segurança que podem diferir de um país para outro, levando em conta as leis existentes da localidade.

2.1.2 Vulnerabilidades em aplicativos *web*

Vulnerabilidades, ou falhas de segurança de *softwares*, são resultados de possíveis erros de projetos, implementações e configurações do sistema, devido à falta de mão de obra qualificada, atualizações mal feitas e muitas vezes a falta de implementação de segurança para *softwares*. São falhas que podem se enquadrar à leis e normas citadas anteriormente e prejudicar o empreendedor, entidade e o cliente final. Essas falhas podem ser exploradas por atacantes, resultando em uma violação da segurança do sistema *web* (FRANZINI, 2009), assim como o demonstrado na Figura 1, onde a ameaça ataca uma aplicação *web*, até que encontre, ao menos, uma falha de segurança, tendo livre acesso pelos controles de segurança da aplicação, caso a aplicação contenha algum, para então ter acessos técnicos aos dados da aplicação e impactar negativamente no negócio empresarial, com vazamentos de dados, ou usufruir dos mesmos.

Figura 1 – Demonstrativo ataque à falha de segurança *web*



Fonte: OWASP (2017, p. 7).

Segundo a OWASP (2017), as dez maiores vulnerabilidades, em ordem crescente, e suas descrições de aplicações *web* estão abaixo.

Injeção de código – Falhas de injeção, tais como injeções de SQL, OS e LDAP, que consistem em introduzir dados ao sistema, para tentativa de invadi-los. Essa é a

vulnerabilidade mais explorada, devido ao fato de ser facilmente detectada por *softwares* de ataques, o que ajuda os atacantes a encontrar as falhas facilmente, e atacar os sistemas com dados hostis, levando-o a executar comandos não pretendidos.

Quebra de autenticação e gerenciamento de sessão – Nessa vulnerabilidade, os atacantes se aproveitam de falhas de autenticação ou sessão de páginas *web*. Desenvolvedores podem deixar expostos esses dados, ao qual acabam permitindo a captura desses dados por programas atacantes que se encontram no computador do cliente.

Exploração de dados sensíveis – Dados sensíveis são os dados pessoais, cartões de crédito, registro de saúde, credenciais, etc. Nesse ataque, os atacantes procuram meios de ataques roubando a chave de segurança, fazendo ataques do tipo *man in the middle*, ou seja, roubar dados ou textos apagados do servidor, enquanto estão navegando, ou nos temporários do navegador do usuário. Geralmente, o atacante só obtém sucesso quando os dados não são criptografados.

Entidades externas de XML (XXE) – Esta é uma vulnerabilidade mais recente, devido a inúmeras utilizações de arquivos com extensão XML. De forma em que estes arquivos XML contêm dados pessoais, muitas vezes, e se mal protegidos, podem ser interpretados por atacantes. Os atacantes utilizam serviços de entidades externas com processador de URI de ficheiros, partilhas internas de ficheiros, pesquisa de portas de comunicação internas, execução de código remoto e ataques de negação, conhecido como *Billion Laughs*.

Quebra de controlo de acessos – Vulnerabilidade de aplicação, uma falha de desenvolvimento, onde a aplicação não verifica os privilégios do usuário no acesso de uma URL, caminho específico do sistema, permitindo o acesso de um usuário não autorizado em uma determinada página do sistema, ao qual esse usuário não deveria ter acesso. Essa configuração pode ser feita através de gerenciamento de outro sistema, onde acaba por vezes também sendo mal configurado e permitindo o acesso indevido.

Configurações de segurança incorretas – Essa é uma vulnerabilidade onde o atacante acessa serviços com seguranças padrões, como por exemplo, um usuário e senha padrão de um banco de dados ou serviço que vem pré-configurado de

instalação. Configurações que acabam passando por padrões de instalação na hora da aplicação, onde o atacante tem acesso fácil para roubar as informações.

Cross-Site Scripting (XSS) – Uma técnica em que o atacante utiliza as falhas de segurança em aplicações que possuem dados não-confiáveis em uma página *web*, ou seja, sem a validação ou filtragem dos mesmos. Normalmente, esse ataque está relacionado ao uso de API com JavaScript, onde o atacante utiliza outra aplicação *web* para enviar códigos maliciosos, geralmente via *script* no *browser* do cliente. Falhas desse tipo são fáceis de serem detectadas por profissionais da área, mas podem passar facilmente por usuários finais de sistemas *web*. Essas falhas normalmente apresentam uma página de *login*, mesmo depois de já ter efetuado o mesmo, ou ainda desfigurar sites, inserir códigos maliciosos, redirecionar usuários a outras páginas, sequestrar dados do navegador utilizando *malwares*, entre outros ataques em cima do navegador do cliente.

Desserialização insegura – Esse é um ataque interno, onde um usuário autorizado muda um valor de parâmetro para outro valor, tentando obter acesso a dados que não tem permissão. Muitas aplicações *web* utilizam um nome ou chave para gerar outras páginas *web*, porém essas aplicações, muitas vezes, não verificam se o usuário está autorizado a acessar tais informações.

Utilização de componentes vulneráveis – Alguns componentes de desenvolvimento *web*, como bibliotecas e *frameworks*, podem conter vulnerabilidades e serem encontradas por atacantes. Dessa forma, o atacante pode explorar essa vulnerabilidade por meio de *exploit*, um *software* de ataque, ou um pacote de transmissão para invadir o sistema (PRADA, 2008). É um erro grave na maioria das aplicações, devido aos desenvolvedores não manterem atualizado as bibliotecas utilizadas.

Registro e monitorização insuficiente – A falta de monitoramento das redes e sistemas, permite aos atacantes abusar do sistema, por terem o tempo necessário de utilizar a força bruta persistente, até conseguir invadir o sistema. Esses ataques não monitorados internamente, demonstram um tempo necessário de duzentos dias para serem detectados, e normalmente são detectados por entidades externas, ao invés de monitoramentos internos da empresa.

A Figura 2 apresentada pela OWASP (2017), com os níveis de dificuldade em

uma escala de 1 a 3, sendo 3 o mais prejudicial e 1 menos prejudicial, o instituto demonstra e avalia as dificuldades do atacante, de se infiltrar no aplicativo. Seguido pela prevalência, a quantidade de vulnerabilidades disponíveis para o atacante. Seguido pela detecção, nível de dificuldade para detectar o ataque. Seguido pelo impacto técnico, nível de dados não autorizados que o atacante terá. Seguido pelo escore, nível de falhas apresentadas em sistemas avaliados pela OWASP (2017), conforme Figura 2.

Figura 2 – Resumo dos riscos do top 10 *OWASP*

RISCO	Agente Ameaça	Vetores Ataque	Falha Segurança		Impacto		Score
		Abuso	Prevalência	Deteção	Técnico	Negócio	
A1:2017-Injeção	Específico App.	FÁCIL: 3	COMUM: 2	FÁCIL: 3	GRAVE: 3	Específico App.	8.0
A2:2017-Quebra de Autenticação	Específico App.	FÁCIL: 3	COMUM: 2	MODERADO: 2	GRAVE: 3	Específico App.	7.0
A3:2017-Exposição de Dados Sensíveis	Específico App.	MODERADO: 2	PREDOMINANTE: 3	MODERADO: 2	GRAVE: 3	Específico App.	7.0
A4:2017-Entidades Externas de XML (XXE)	Específico App.	MODERADO: 2	COMUM: 2	FÁCIL: 3	GRAVE: 3	Específico App.	7.0
A5:2017-Quebra de Controlo de Acessos	Específico App.	MODERADO: 2	COMUM: 2	MODERADO: 2	GRAVE: 3	Específico App.	6.0
A6:2017-Configurações de Segurança Incorretas	Específico App.	FÁCIL: 3	PREDOMINANTE: 3	FÁCIL: 3	MODERADO: 2	Específico App.	6.0
A7:2017-Cross-Site Scripting (XSS)	Específico App.	FÁCIL: 3	PREDOMINANTE: 3	FÁCIL: 3	MODERADO: 2	Específico App.	6.0
A8:2017-Desserialização Insegura	Específico App.	DIFÍCIL: 1	COMUM: 2	MODERADO: 2	GRAVE: 3	Específico App.	5.0
A9:2017-Utilização de Componentes Vulneráveis	Específico App.	MODERADO: 2	PREDOMINANTE: 3	MODERADO: 2	MODERADO: 2	Específico App.	4.7
A10:2017-Registo e Monitorização Insuficiente	Específico App.	MODERADO: 2	PREDOMINANTE: 3	DIFÍCIL: 1	MODERADO: 2	Específico App.	4.0

Fonte: OWASP (2017, p. 24).

Ainda segundo a instituição OWASP (2017), as vulnerabilidades citadas acima, que podem ser de impacto grave, conforme visto na Figura 2, podem ser corrigidas, mitigadas e minimizadas com a preocupação de segurança do desenvolvimento de *software* desde o início. Para isso, a empresa de desenvolvimento deve seguir algumas boas práticas, que podem ser citadas em documentos de guia rápido, com embasamento de estudos realizado pela própria instituição.

2.1.3 Leis de proteção de dados para aplicativos

Toda aplicação de *software* no Brasil está sujeita a normas de leis brasileiras. O país tem hoje cinco leis de desenvolvimento desses aplicativos, sendo que em setembro de 2020 entrou em vigor a Lei Geral de Proteção de Dados. Seguem as descrições:

- Lei de proteção da propriedade intelectual de programas de computador (Lei nº 9.609, de 19 de fevereiro de 1998).

Esta lei trata sobre os direitos autorais do desenvolvedor, onde o programador do *software* pode exigir uma remuneração de *royalties* ou faturamento entre acordo pelo código fonte utilizado em um sistema ou aplicação. Porém, é fundamental que o sistema esteja registrado. Os direitos de tutela do código são válidos por cinquenta anos desde o seu registro.

- Lei de propriedade industrial (Lei nº 9.270, de 14 de maio de 1996).

No desenvolvimento de uma aplicação, a marca, nome da aplicação é sempre muito fundamentada, essa por sua vez está protegida juntamente com a marca da empresa, pela qual é regida pela Lei nº 9.270, que regula os direitos e obrigações relativos à propriedade industrial.

- Código de defesa do consumidor (Lei nº 8.078, de 11 de setembro de 1980).

A mesma Lei que engloba o comércio brasileiro, também define os valores do consumidor diante de uma aplicação *web*. Dentre as regras de lei do consumidor, assim como em outros comércios, valem para aplicações de *software*, onde o sistema deve cumprir com o tratado, podendo ser questionado caso não haja uma transparência no descritivo do anúncio.

- *E-commerce* (Decreto 7.962, de 15 de março de 2013).

Este decreto é uma correção da Lei nº 8.078, que não previa regras próprias para os termos de negociação por *software*. Decretando que toda a negociação *E-Commerce* deve conter os dados do fornecedor, em visualização fácil e de destaque. Além disso, o decreto prevê que o consumidor poderá cancelar ou devolver a compra em até 7 dias, após a entrega do produto, sem que o consumidor tenha que justificar a devolução.

- Marco civil da internet (Lei nº 12.965, de 22 de abril de 2014).

O Marco Civil trata a *internet* como um uso fundamental para liberdade de expressão. A Lei define que “o acesso à *internet* é essencial ao exercício da cidadania”. Ao mesmo tempo, a Lei define que o usuário tem sua vida privada não violada, onde também consta que a qualidade de conexão da *internet* não pode ser vendida conforme o uso do cliente, e que os dados do usuário só poderão ser repassados para terceiros se assim liberados pelo usuário, ou em casos judiciais.

Outro ponto importante do marco civil é que o provedor de *internet* não se responsabiliza pelos dados postados ou adquiridos pelos seus clientes, mas empresas com serviços de redes sociais, vídeos e outros podem ser responsabilizados, e responder judicialmente caso não retirem os materiais assim pedidos judicialmente.

- Lei geral de proteção de dados – LGPD (Lei nº 13.709, de 14 de agosto de 2018).

A LGPD é a Lei Nº 13.709, aprovada em agosto de 2018 e com vigor em 18 de setembro de 2020. Esta, por sua vez, é uma lei que cria um cenário de segurança jurídica, juntamente com normas e práticas promovendo a proteção dos dados dos clientes.

É importante citar que o vigor da lei independe de onde os dados estão armazenados, se for em território nacional ou internacional, a lei se aplica igualmente. Podemos resumir brevemente a lei em:

- Consentimento: onde o cliente deve ter o consentimento dos seus dados serem armazenados ou tratados. Podendo ser tratado sem consentimento apenas se isso for indispensável para cumprir uma obrigação legal.
- Automatização com autorização: mais uma garantia do cliente, onde ele tem o direito de solicitação para deletar seus dados do sistema, revogar um consentimento, transferir dados ou outros tratamentos de seus dados.
- Autoridade Nacional de Proteção de Dados Pessoais (ANPD): este é o órgão público de consentimento, aplicação e fiscalização para que a lei seja cumprida, caso não seja cumprida, o mesmo órgão público poderá penalizar os responsáveis. Para aumentar a fiscalização e não depender apenas do órgão público, a lei estipula os

agentes do tratamento de dados. Nas organizações, tem o controlador, que reúne as informações e interage com o cliente e as autoridades, e o operador, o qual realiza o tratamento dos dados.

- Gestão em foco: o item da lei que mais tem a ver com a informática, pois nele está descrito que a administração de riscos e falhas deve ser gerido por quem gerencia as bases de dados, adotando medidas de segurança preventivas, boas práticas e certificações existentes no mercado. A entidade responsável também deve elaborar planos de contingência, auditoria dos dados, e resolver incidentes com agilidade.

Em caso de vazamento dos dados, a ANPD e os indivíduos afetados devem ser avisados imediatamente. Lembrando que todos os agentes de tratamento se sujeitam à lei. Isso significa que a organização e os responsáveis respondem pelos danos causados, onde uma falha de segurança pode gerar multas de 2% do faturamento anual da organização, com limite de R$ 50 milhões por infração (LGPD, Lei nº 13.709).

2.1.4 Melhores práticas de programação segura

O uso de um *WAF* não resolve as vulnerabilidades, apenas minimiza as mesmas, por isso a empresa de desenvolvimento de *software* deve seguir normas e boas práticas de segurança. Segundo a OWASP (2012), o desenvolvimento de *software* para *web* deve seguir normas de segurança e boas práticas, logo, a instituição OWASP (2012) cita algumas boas práticas em seu guia rápido, são elas:

- Validação dos dados de entradas;
- Codificação dos dados de saída;
- Autenticação e gestão de credenciais;
- Gestão de sessões;
- Controle de acessos;
- Práticas de criptografia;
- Tratamento de erros e *logs*;
- Proteção de dados;

- Segurança na comunicação;
- Configurações do sistema;
- Segurança em base de dados;
- Gestão de arquivos;
- Práticas gerais de sistema.

Ainda segundo a instituição OWASP (2012), o objetivo da segurança em aplicações é manter a confidencialidade, integridade e disponibilidade, já citados anteriormente.

Para a instituição OWASP (2012), é mais barato desenvolver um *software* seguro desde o início, do que corrigir os problemas posteriormente. Para tal finalidade, a instituição define o uso das boas práticas citadas acima, juntamente com ferramentas de análises *Penetration Test (Pentest)*, próximo capítulo, e o uso de ferramentas que possam mitigar os ataques e minimizar a vulnerabilidades do sistema. De forma que possa reduzir processos de rede, que causam e apresentam lentidão à rede, assim como proteger o servidor e as aplicações dos ataques.

A equipe da Gocache (2017) apresenta um guia de segurança para a implementação de aplicações *web*, mesmo com a implementação de um *WAF* junto, ressaltam outras seguranças que devem ser seguidas. A equipe ressalta em sua lista:

- Criar um plano de segurança para aplicações *web*;
- Realizar um inventário das suas aplicações *web*;
- Classifique as aplicações em três categorias: crítico, importante e normal;
- Priorizar vulnerabilidades;
- Executar aplicativos usando o mínimo de privilégios;
- Proteja-se mesmo durante o processo de avaliação utilizando um *WAF*;
- Use *cookies* de forma segura;
- Implementar as seguintes sugestões de segurança *web*: utilizar SSL, cabeçalho de segurança, utilizar políticas de segurança e senhas com segurança definidas como forte;
- Conduzir treinamentos e conscientização de segurança em aplicações *web*.

2.1.5 *Pentest* em aplicativos *web*

Um dos métodos de avaliação de vulnerabilidades em aplicativos *web*, é pelo teste de *Penetration Test (Pentest),* este é um teste realizado de forma controlada, onde o atacante é contratado para realizar testes de vulnerabilidades com o total conhecimento da entidade final e da empresa de *software*.

Para elaborar um *Pentest*, o profissional especializado precisa seguir algumas normas, encontrando ou não vulnerabilidades no sistema, o profissional deve manter todos seus passos e análises registrados, para um eventual estudo e mitigação da vulnerabilidade, caso ocorra. O *Pentest* pode ser separado em 5 fases de testes (WILHELM, 2013).

Reconhecimento e identificação do alvo – Esta é a fase inicial de um ataque, onde o profissional da área reúne todas as informações possíveis sobre o sistema, inicialmente de forma passiva, conhecendo o alvo que irá atacar. Após isso, o profissional obtém parâmetros de ataque de forma ativa, para um reconhecimento de possíveis ataques em portas de conexão de rede, IP de acesso, sistemas instalados, e toda informação que conseguir reunir sem utilizar força bruta.

Reconhecimento das vulnerabilidades – Esta é a fase que o atacante profissional da área de segurança faz um breve ataque inicial, para reunir mais informações, como portas vulneráveis sem restrições, serviços disponíveis e suas respectivas versões, versões de sistemas operacionais, informações do *hardware*, entre outras informações que o ajudem a chegar a vulnerabilidades maiores.

Ganho de acesso – Esta é a etapa de ataque inicial pelo profissional, onde ele determina o melhor horário para atacar o sistema, de forma que utilize ferramentas que lhe auxiliam ao ataque, com a definição final de possuir controle de um ou mais sistemas instalados no alvo, se possível, obter acesso de administrador do sistema operacional.

Preservação de acesso – Nesta etapa, após conseguir o acesso administrativo do sistema, o mesmo adquire um acesso paralelo e administrativo, o qual é conhecido por *backdoor*, desta forma, o profissional obtém acesso administrativo mesmo que a conexão seja brevemente interrompida. De forma que possa realizar consultas em qualquer nível de hierarquia de permissão do sistema.

Cobrindo evidências – Na etapa final do ataque ao sistema, o profissional, antes mesmo da desconexão ao alvo, apaga todos seus rastros, deletando *logs* de registros ao sistema, desta forma ele provém que não fique armazenado informações sobre sua conexão, ficando anônimo. O profissional realiza esta tarefa, mesmo que utilize formas de dificultar sua origem de ataque, com servidores *Proxy*, acesso via VPN, entre outros meios.

Lembrando que o ataque profissional deve ser de conhecimento da empresa alvo, e que todos os passos durante o ataque devem ser documentados, sendo efetivo ou não em seus ataques, e quais informações conseguiu obter. Desta forma, posteriormente, a empresa alvo possa tomar as medidas de mitigação e correção das vulnerabilidades encontradas pelo profissional em seu sistema.

Citando ainda outra forma de *Pentest*, que pode ser dividido em três formas de análise sob o alvo de ataque, onde a escolha de ataque pode influenciar nos resultados obtidos (ASSUNÇÃO, 2014). Lembrando que sempre deve ser documentado todo passo de ataque pelo profissional. As três formas de análise são:

- *Black Box* – Este é um ataque às cegas, o atacante profissional não tem informação alguma do que vai atacar, dos sistemas, proteções, informações, ou outros. É realmente dado como um ataque de *hacker* pela internet, procurando um alvo.

- *White Box* – Este é um ataque onde o atacante profissional tem total conhecimento do que está atacando, quais sistemas estão sendo utilizados, quais são as informações pertinentes. Sua utilização é para um nível mais crítico de mitigar as vulnerabilidades possíveis.

- *Grey Box* – Neste ataque temos uma mistura dos dois anteriores, o atacante profissional sabe alguns dados sobre o sistema e empresa, e tenta conseguir mais informações, atacando de formas menos aleatórias, sabendo algumas informações básicas. Utilizado para mitigar ataques de usuários ou infiltrados na rede.

2.2 *Firewall*

Segundo Ford (2002), *firewalls* são classificados como dispositivos ou programas projetados para detectar e proteger computadores e redes contra ameaças internas e externas. O autor ainda classifica *firewall* em dois modelos, *firewall* de *software* e *firewall* de *hardware*. Ambos *firewalls* possuem a mesma descrição de segurança, detectar ameaças e proteger o computador local ou a rede.

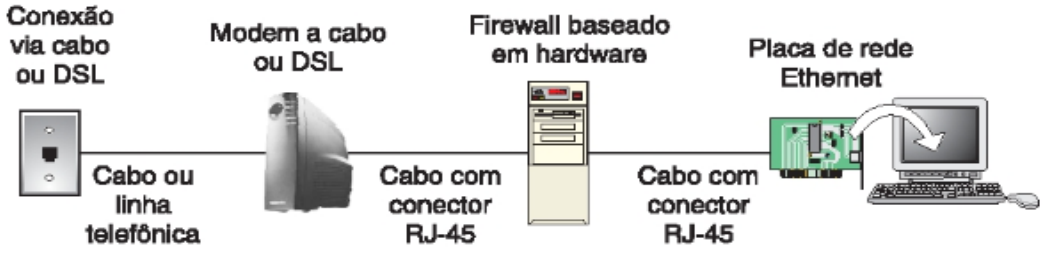
Neste capítulo, demonstraremos um pouco de *firewall* e algumas ferramentas de percepção de vulnerabilidade em pacotes de dados que passam em *firewalls*, e suas análises para decidir as vulnerabilidades e a decisão tomada pelas próprias ferramentas do que acontece com o pacote de dados de rede com ou sem vulnerabilidades.

2.2.1 Como funciona um *firewall*

Firewalls baseados em *hardware* são dispositivos que atuam na conexão da *internet* entre o computador do usuário e a *internet*, isso através de um navegador *web*, sem ser necessário outro *software* adicional (FORD, 2002).

Na Figura 3, podemos verificar a posição de implementação de um *firewall* por *hardware*, primeiramente a *internet* é disponibilizada por uma com conexão via cabo ou DSL, à qual é conectada ao modem a cabo ou DSL para se dar a conexão ao provedor de *internet*. A seguir, é implementado um *firewall* baseado em *hardware*, esse que, por sua vez, fará toda a segurança da rede, podendo ser conectado a um único computador, conforme a Figura 3, ou a um *switch*, distribuindo a segurança para todos os computadores de uma rede.

Figura 3 – Uma configuração de *firewall* baseado em *hardware*



Fonte: Ford (2002, p. 41).

O *firewall* demonstrado por Ford (2002) aplica-se a uma rede de dados, para assegurar os acessos externos e proporcionar um mínimo de segurança à rede interna. Este modelo de *firewall* minimiza a chance de um ataque a sua rede estrutural, baseada em TCP/IP e *MAC*.

O *firewall* mostrado acima é uma segurança necessária, mas não protege o sistema de aplicação *web*, ele não é definido a esta função. Para tal aplicação de segurança, existem ferramentas específicas, assim nomeadas de *firewall* para aplicações *web* (KUROSE; ROSS, 2014).

2.2.2 O que é IDS e IPS

Para uma segurança da rede e das aplicações, não basta manter os *softwares* atualizados, usar pacotes de correções, ou ter um *firewall* citado anteriormente. Os especialistas na área de segurança sabem a necessidade de implementação de ferramentas de monitoramento, como, por exemplo, ferramentas determinantes na descoberta e prevenção de intrusão, que podem ser classificados em IDS e IPS (GALVÃO, 2015).

Conforme Galvão (2015), *Intrusion Detection System* (IDS) ou Sistema de Detecção de Intrusões, é uma ferramenta que analisa os dados que trafegam na rede, comparando-os com assinaturas de ataques previamente preparados, sendo possível identificar qualquer tipo de vulnerabilidade de ataque que possa ocorrer na rede.

Ainda segundo Galvão (2015), *Intrusion Prevention System* (IPS) ou Sistema de Prevenção de Intrusões, é o complemento de IDS. Sua função é prevenir intrusões juntamente com IDS e bloquear o tráfego desses ataques, resolvendo a vulnerabilidade antes da mesma causar algum impacto.

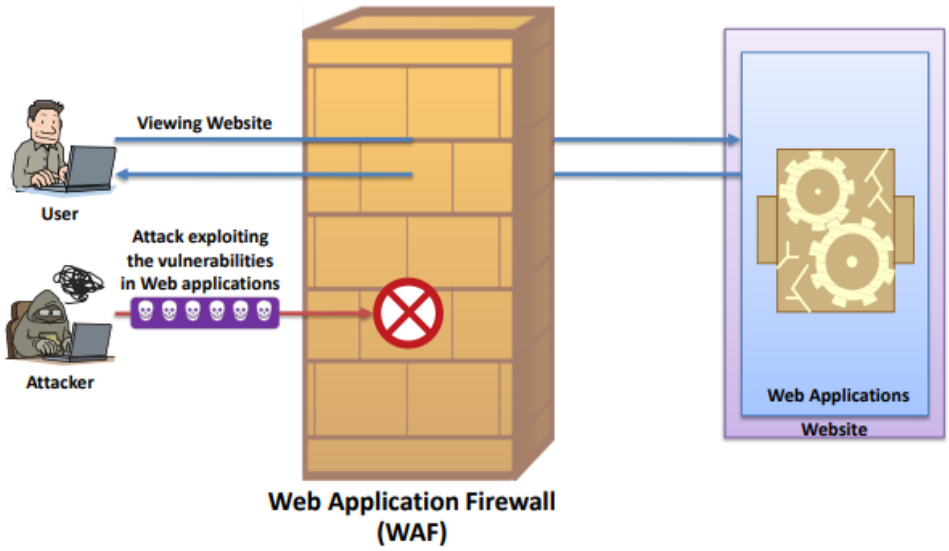
2.2.3 Conceitos de *firewall* para aplicativos *web*

Outra ferramenta que auxilia a minimizar as vulnerabilidades de uma aplicação é o conceito de *WAF*, dado por sua definição como um *hardware* ou *software* que protege uma ou mais aplicações *web* de ataques que exploram vulnerabilidades de aplicações *web* (IPA, 2011).

A função do *WAF* é verificar os pacotes de transação entre aplicação e cliente, como também entre cliente e aplicação, em busca de vulnerabilidades nas regras declaradas em sua base de dados. Desta forma, podemos verificar que o *WAF* pode conter brechas de segurança se não contiver as regras de vulnerabilidades, por isso, ela é dada como a implementação que minimiza a vulnerabilidade, e não elimina completamente ataques às aplicações *web*. Observa-se na Figura 4 a aplicação de um *WAF* com um ataque em andamento (IPA, 2011).

A Figura 4 demonstra uma aplicação de *WAF* em uma rede, de forma que dois usuários estejam usando a aplicação *web* no mesmo instante, onde o usuário legítimo consegue trafegar pelo *WAF*, devido seus pacotes de transação não conterem nenhuma vulnerabilidade, e ter acesso à aplicação *web*. Já o usuário malicioso, cai nas regras de vulnerabilidades do *WAF,* e acaba sendo *dropada* a conexão com a aplicação *web*, esta atitude pode ser configurada no *WAF*, onde a conexão pode ser *dropada* ou pode retornar algum erro ao usuário malicioso, como uma página com o erro 403 (*Forbidden*) (IPA, 2011).

Figura 4 – Funcionamento de *WAF*



Fonte: IPA (2011, p. 9).

2.2.4 Diferença entre *firewall*, IDS, IPS e *WAF*

Nos capítulos anteriores é definido algumas ferramentas utilizadas para assegurar a rede e aplicativos *web*. Cada uma com suas peculiaridades e sua posição

de implementação. Neste capítulo, vamos diferenciar elas com suas vulnerabilidades em segurança.

Conforme apresentado anteriormente, segundo Ford (2002), o *firewall* tradicional é o controle de acessos de uma rede, restringindo o usuário a um destino único, criando apenas um canal de acesso, de forma a assegurar parte da rede. Este acesso é restrito por protocolos TCP/IP, juntamente com o destino das portas de acesso de cada protocolo. Ainda segundo Ford (2002), um *firewall* não protege o destino final do acesso, apenas centraliza todo acesso a um único serviço, definido pela porta de acesso e destino interno. Isso significa que ataques presentes no *payload* do pacote não serão identificados por este tipo de filtro (NAKAMURA; GEUS, 2007).

O *firewall* citado no parágrafo anterior pode ser auxiliado para minimizar as vulnerabilidades utilizando outras ferramentas citadas em capítulos anteriores, são elas IPS/IDS e *WAF*. As ferramentas IPS e IDS, juntas, podem prevenir ataques generalizados, ou seja, ataques de qualquer pacote de dados na rede, de forma que o IPS bloqueia essas vulnerabilidades se baseando em uma *blacklist,* uma lista de regras que define valores inaceitáveis para esta rede (IPA, 2011).

A diferença entre IDS e IPS é de que o IDS é uma ferramenta apenas para detecção de intrusão ou ataque, enquanto o IPS detecta um ataque e pode acabar com a vulnerabilidade do ataque. Desta forma, a ferramenta IPS é posicionada de forma *inline*, onde os pacotes da rede são obrigados a passar pela ferramenta e serem analisadas. A ferramenta IDS pode ser tratada e posicionada de forma *inline* ou posicionado de forma passiva na rede, onde é criado uma cópia dos pacotes destinados a rede, para então serem avaliados (NAKAMURA; GEUS, 2007).

Conforme citado anteriormente, um *WAF* é um *hardware* ou *software*, cuja funcionalidade é analisar os pacotes de transição na rede, avaliando cada pacote com um parâmetro existente em uma lista. Mas diferentemente dos protocolos anteriormente, o *WAF* possui uma *blacklist* e possivelmente uma *whitelist*, sendo mais efetivo também em sua *blacklist*, de forma que sua especialidade é apenas ser uma ferramenta de segurança para aplicações *web*, negando ou *dropando* os pacotes que contêm vulnerabilidades existentes na *blacklist*. Já sua *whitelist* atua de forma

contrária a *blacklist*, deixando passar apenas os comandos citados na lista, negando ou *dropando* todo o resto dos pacotes (IPA, 2011).

Figura 5 – Comparação *firewall*, IPS e *WAF*

	Firewall	IPS	WAF
Controle de acesso	Algum	Algum	Sim
Detecção de protocolo HTTP	Algum	Sim	Sim
Bloqueio de DoS (sob HTTP)	Algum	Algum	Algum
Identificação de ataques Web	Não	Algum	Sim
Detecta ataques em sessão HTTP (múltiplas requisições)	Não	Não	Sim
Suporte a tráfego criptografado (SSL)	Não	Não	Sim
Monitoração de erros do servidor web	Não	Não	Sim
Transcrição das sessões http(s)	Não	Algum	Sim

Fonte: Silveira (2012, p. 6).

Na Figura 5 temos a real comparação de um *firewall*, um IPS e um *WAF*, de forma avaliativa dos recursos disponíveis em cada ferramenta. Podemos notar que uma aplicação *web* com IPS é menos vulnerável do que apenas o uso de um *firewall* tradicional, tendo como segurança a mais do uso de um IPS a detecção de protocolo HTTP, Identificação de ataques *web* e transcrição das sessões HTTP/HTTPS. Mas o uso de uma ferramenta *WAF* pode ser menos vulnerável do que um IPS com total controle de acesso, total identificação de ataques *web*, detecção de ataques em sessões HTTP, suporte a tráfego criptografado (SSL), monitoração de erros do servidor *web* e total transcrição das sessões HTTP/HTTPS. Desta forma, é imprescindível o uso de uma ferramenta *WAF* para a segurança da aplicação *web*.

2.3 Filtro avançado para aplicativos *web*

Este capítulo tem por objetivo exemplificar o estudo de um *WAF*, o tema principal deste trabalho. Entretanto, o assunto de *WAF* é demasiado extenso em sua abrangência na segurança, nas definições e em seus conceitos para um estudo de profundidade. Portanto, serão abordados apenas estudos essenciais para o

entendimento da aplicação final do *WAF* projetado neste trabalho.

2.3.1 *Blacklist* e *Whitelist*

O *WAF* utiliza listas predefinidas para analisar os pacotes interceptados antes de chegar na aplicação *web*, esta lista pode ser dada por *blacklist*, lista predefinida que contém as vulnerabilidades que serão bloqueadas, dessa forma, o pacote de dados é descartado, evitando ataques a aplicação *web*. Ainda pode ser dada como *whitelist*, predefinida como uma lista que permita apenas as operações que constam na lista, negando todas outras operações (IPA, 2011).

Segundo IPA (2011), a *blacklist* é definida por uma lista de regras com valores padrões inaceitáveis para transações HTTP. Dessa forma, ao inspecionar uma requisição, o *WAF* procura pelos valores constantes na lista, caso seja dado como requisição maliciosa, é negado ou *dropado* o pacote, caso contrário, é considerada uma requisição normal.

Ainda segundo IPA (2011), a *whitelist* é definida por uma lista de regras com valores padrões permitidos e aceitáveis para transações HTTP. Dessa forma, se o pacote de dados presente em uma requisição HTTP não corresponder com os valores constantes na lista, é caracterizada como maliciosa e negada ou *dropada* a transação. Caso contrário, a transação é dita normal.

2.3.2 Tipos de *Web Application Filter*

Existem alguns modelos de *WAF* com diferenças de implementação, porém a mesma aplicação final, um *WAF* pode ser executado como aplicação de rede local, *plug-in* de servidor ou serviço na nuvem. Cada modelo de implementação possui as vantagens para uma determinada função (EVEO, 2018).

WAFs de rede – esse modelo é implementado localmente, normalmente em um *hardware* próprio, e tende a ser o mais rápido dentre os três modelos disponíveis. Seu gerenciamento é oferecido como um serviço, o que torna sua programação mais simples, de fácil modificação de assinaturas e configurações, o que o torna disponível para atender vários aplicativos com menos esforço. O ponto negativo, é que estes *WAFs* de rede geram altos custos às empresas, pois necessitam um *hardware* e manutenção, juntamente com outras dependências, como alta disponibilidade (EVEO,

2018).

WAFs de host – este modelo de *WAF* possui uma grande vantagem de ter a possibilidade de incluir opções de personalização a um custo baixo, pois sendo totalmente baseado em *software*, ele pode compartilhar o *hardware* com outras aplicações. Mas pode ser uma tarefa difícil configurar e manter um *WAF* compartilhado com outras aplicações, pois eles demandam de bibliotecas locais, como *Java* ou .NET, e são dependentes de recursos de servidores locais, que são compartilhados, e podem criar gargalos na sua eficiência (EVEO, 2018).

*WAF*s na nuvem – este modelo de *WAF* é hospedado em nuvens, normalmente administrados pelos provedores do serviço que disponibilizam a interface. Estes *WAFs* normalmente já vem pré-configurados para uma aplicação, desta forma o provedor pode utilizar o mesmo recurso de *WAF* para mais aplicações em comuns, escalonando e baixando o custo (EVEO, 2018).

2.3.3 Ferramenta *ModSecurity*

Definido como uma ferramenta para monitoramento, registro e controle de acesso em tempo real de aplicativos *web*, a ferramenta *ModSecurity* é compatível com *Unix, Linux, Windows e Macintosh*. O *ModSecurity* é um facilitador, uma ferramenta *WAF* de *host* programável, onde o programador pode escolher os caminhos através dos recursos disponíveis. Essa liberdade de escolha em sua lógica de atuação como *WAF* é que o torna uma biblioteca de código aberto. Dessa forma, o *ModSecurity* disponibiliza acesso total ao código-fonte, onde o programador pode ajustar a ferramenta às suas necessidades, tornando-o prático (FOLINI; RISTIC, 2018).

Segundo Folini e Ristic (2018), o *ModSecurity* pode ser separado em uma lista de cenários importantes:

- Monitoramento de segurança de aplicativos em tempo real e controle de acesso:

Folini e Ristic (2018), destacam ainda que o *ModSecurity* fornece acesso ao tráfego HTTP/HTTPS em tempo real para poder inspecioná-lo no exato momento. Ainda nessa análise, o *ModSecurity* armazena os resultados em um *buffer*, podendo executar correlações de pontuação de eventos com as regras de segurança, que

podem ser definidas para bloquear ou liberar, junto a uma solicitação, onde o analista por trás do *ModSecurity* pode liberar ou não o acesso que se encontra nesse *buffer*.

- Registro de tráfego HTTP/HTTPS completo:

Ainda segundo Folini e Ristic (2018), os servidores de *web* não armazenam *logs* para fins de segurança, registram muito pouco os acessos e ajustes de transação no serviço *web*. O *ModSecurity* oferece a capacidade de registrar tudo que chegue ao serviço *web* do servidor, incluindo dados brutos de transação, essenciais para uma perícia. Ainda, na ferramenta é possível escolher as transações registradas e quais partes serão registradas.

- Avaliação de segurança passiva contínua:

Essa avaliação é um evento agendado e programado. Normalmente, é um evento para testes de vulnerabilidades em tempo real, onde o *ModSecurity* se concentra em comportamentos do próprio sistema, ao invés de acessos externos (FOLINI; RISTIC, 2018).

- Proteção de aplicativo da *web*:

Ainda segundo Folini e Ristic (2018), o *ModSecurity* reduz as vulnerabilidades, restringindo seletivamente os recursos HTTP/HTTPS (métodos de solicitação, cabeçalhos de solicitação, tipos de conteúdo, etc.). O *ModSecurity* funciona em paralelo com módulos *Apache*, desta forma, impõe muitas restrições semelhantes, assim como é possível corrigir problemas de gerenciamento de sessões, bem como falsificação de solicitações.

- Flexibilidade de programação:

Folini e Ristic (2018), reforçam que o *ModSecurity* possui uma flexibilidade em segurança, onde o programador pode escolher a maneira de atuação do *WAF*. Mas também pode ser utilizado como roteador de serviços *web* XML, combinando a capacidade de analisar XML e aplicar expressões definidas.

2.3.4 *OWASP ModSecurity Core Rule Set* (CRS)

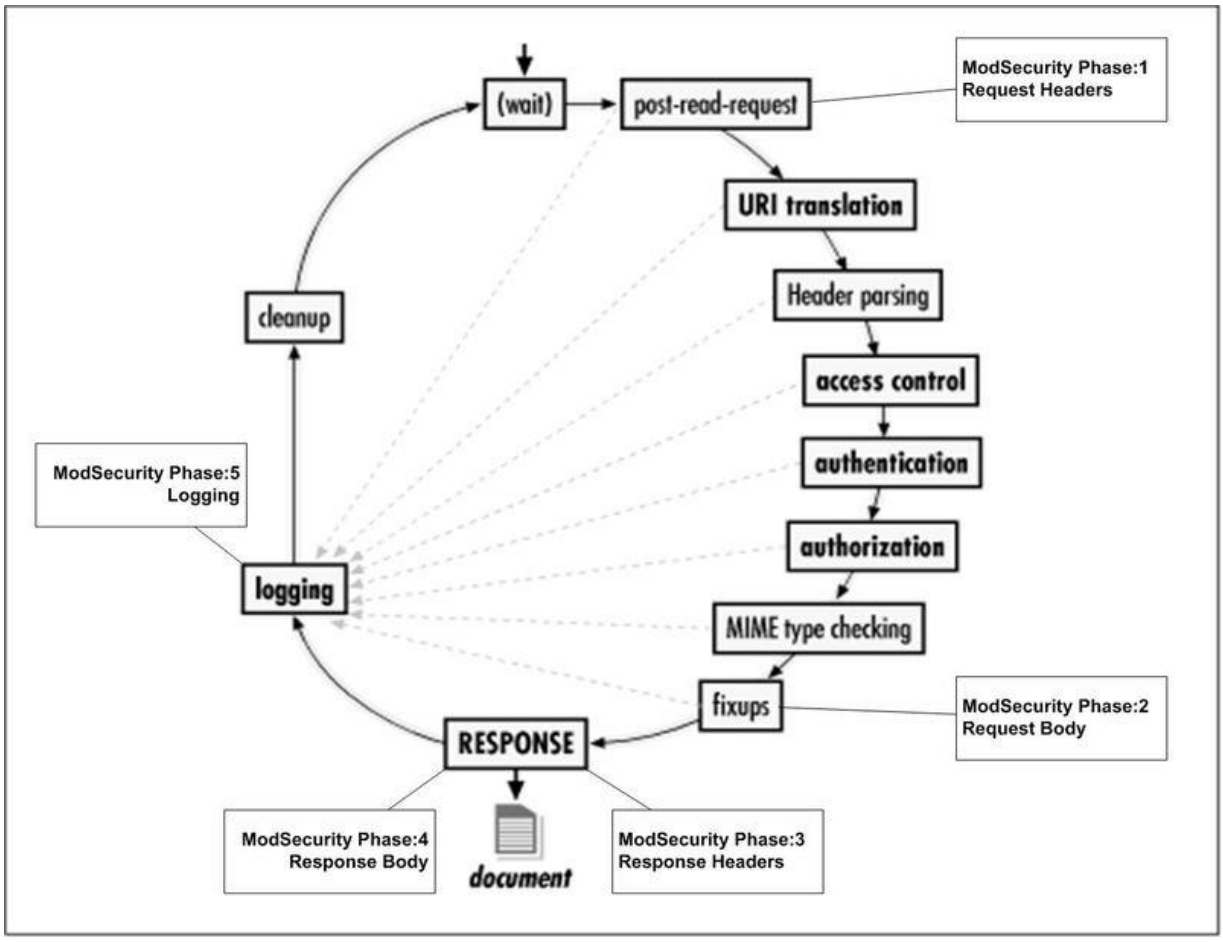
O objetivo do projeto de regras para *ModSecurity*, juntamente com o instituto *OWASP*, é de fornecer um conjunto fácil de regras para vulnerabilidades e ataques, que minimize ao máximo as vulnerabilidades e ataques para aplicativos *web*. A versão mais recente 3.2 desta união, fornece as seguintes proteções (FOLINI, 2019).

- Proteção de protocolo HTTP;
- Pesquisas na *blacklist* em tempo real;
- Proteções de negação de serviço HTTP;
- Proteção genérica contra ataques na *web*;
- Detecção e ocultação de erros.

2.3.5 Como funciona o *ModSecurity*

Segundo a Malware Expert (2017), a ferramenta *ModSecurity* pode ser dividida em cinco fases, conforme a Figura 6.

Figura 6 – Funcionamento *ModSecurity*



Fonte: Malware Expert (2017).

Fase 1 – Requisição de cabeçalho.

A fase inicial do *ModSecurity* acontece antes do *Apache* receber a solicitação do usuário. Nesta fase, o *WAF* analisa os cabeçalhos de solicitação HTTP/HTTPS.

Nesta fase, o *ModSecurity* determina o tipo de análise a ser processado, por exemplo, se for um XML ou não, analisando as autenticações necessárias para transição, conforme Figura 6.

Fase 2 – Requisição de corpo.

Nesta fase, o *ModSecurity* analisa o corpo da solicitação. Ainda nesta fase, o *Apache* recebe a solicitação e o *ModSecurity* determina um modelo de técnica de codificação, dentre três, para análise. Podendo ser as técnicas de transferência de dados do formulário, transferência de arquivos ou para dados XML, essas por sua vez analisadas pelo *ModSecurity*, que compara com as regras pré-definidas.

Fase 3 – Resposta do cabeçalho.

Essa é a fase em que o servidor começa a responder e enviar os dados de solicitação de volta ao usuário. Nesta fase, assim como na fase inicial, o *WAF* analisa o cabeçalho de retorno antes de enviar ao usuário.

Fase 4 – Resposta do corpo.

Nesta fase, as regras são implementadas para analisar o conteúdo da transição, podendo analisar respostas como falhas de autenticação ou mensagem de erro.

Fase 5 – Conectando-se.

Na última fase, o *ModSecurity* aplica novas regras de vulnerabilidades e ataques, antes de aplicar os registros no *Apache*. Após o registro no *Apache*, o *ModSecurity* analisa as mensagens geradas pelo servidor *web Apache*. Nesta fase não é possível bloquear a conexão, pois ela é posterior às fases de solicitação e resposta. Mas pode analisar e inspecionar os cabeçalhos que não estão disponíveis nas fases anteriores.

3 TRABALHOS RELACIONADOS

A segurança de informação vem sendo tratada com maior seriedade nas últimas duas décadas, mas nem por isso podemos dizer que era esquecida antes disso. Muitas matérias, artigos e trabalhos acadêmicos foram apresentados durante todo o crescimento da *internet*, desde o seu surgimento. Neste capítulo, vamos avaliar alguns desses trabalhos que tratam de segurança para aplicações *web*.

Dentre as avaliações, podemos citar que os trabalhos utilizam dados de ataques de instituições privadas e governamentais, reconhecidas, como base de dados de estudos do instituto *Open Source Foundation for Application Security* (*OWASP*).

Costa (2017) utiliza um modelo de trabalho com análises das maiores vulnerabilidades em *sites* governamentais, usando como base de vulnerabilidades os dados de *ranking* da *OWASP* (2013) para suas avaliações. Em sua análise ele define as vulnerabilidades em tipos de ataque e tamanho do impacto. Após, ele relaciona várias ferramentas de *scanner* para testes avaliativos de vulnerabilidades e ataques (*Pentest*), que, por sua vez, a análise é separada por *sites* governamentais de cada estado do Brasil. Onde a proposta de avaliação é dada em comparação pela segurança do *site* em conformidade do PIB de cada estado.

Para sua avaliação, ele utiliza a ferramenta de *Scanner* de vulnerabilidades *Skipfish*, este foi escolhido por Costa (2017), devido seus resultados apresentados em encontrar o maior número de vulnerabilidades e de melhor amostragem apresentada dos resultados. Em seus resultados, ele cita que mais de 90% das vulnerabilidades apresentadas foram de encaminhamentos inválidos de URL, em todos os estados analisados, vulnerabilidade de baixo impacto, conforme sua análise.

Também apresenta vulnerabilidades com maior nível de impacto, como portas de serviço de transmissão de arquivos (FTP, porta 21), serviço de acesso remoto seguro (SSH, porta 22) e conexão de área de trabalho remota do *Windows* (RDP, porta 3389). Portas que são alvos recorrentes de ataques com um alto risco de vulnerabilidade e impacto. Em sua conclusão, obteve resultados de que mesmo os estados com maior PIB, apresentam grandes vulnerabilidades de alto impacto, não sendo relacionado o investimento de segurança em aplicações *web* com o PIB do estado.

Hainzenreder (2019) apresenta um trabalho de conclusão, onde relaciona a segurança de aplicações com uma estrutura de Centro de Operações de Segurança (*Security Operation Center - SOC*). Segundo Hainzenreder (2019), um *SOC* deve ser dividido em duas unidades, unidades internas e unidades externas. Internas que, por sua vez, são tratadas por uma equipe interna de monitoramento, aos quais são especialistas na área e cada profissional realizando uma única função, o autor cita exemplos como *Security Administration*, *Incident Response* e *Security Monitoring and Detection*.

As unidades externas, por sua vez, são serviços terceirizados, contratados de forma quando seja necessário, o autor ainda cita exemplos de *pentesting*, *red-teaming* e *threat research*. Hainzenreder (2019) ainda cita hierarquias de equipe de um *SOC*, como, por exemplo, a saída ou troca de departamento de um membro da equipe, as diretivas de segurança destas pessoas devem ser revogadas e atualizadas para o novo departamento.

Em seu trabalho, o autor ainda cita ferramentas de auxílio para análises da equipe, em seu uso, o autor destaca um Sistema de Detecção de Intrusão (IDS), o classificando para o melhor uso da equipe, um Sistema de Prevenção de Intrusão (IPS), ainda destaca o uso de *firewall* e segmentação e segregação de rede. O autor ainda destaca o uso de *Security Information and Event Management* (*SIEM*), uma ferramenta que agrega os dados de eventos produzidos pelos dispositivos da rede, para um melhor monitoramento da equipe, assim como outras ferramentas de análise, em tempo real, destacadas ao longo de seu trabalho.

A relevância de seu trabalho é a citação de um *Firewall* de Aplicações *Web* (*WAF*), nesta abordagem ele cita expressões de segurança implementadas pelo *WAF*.

Sobretudo, ainda considerando as análises de maiores vulnerabilidades de aplicações *web* pelo instituto *OWASP*, base de dados de vulnerabilidades de seu trabalho, correlacionando estes dados com níveis de análise e segurança com o uso de uma equipe *SOC*.

Em sua conclusão, Hainzenreder (2019) corresponde ao uso necessário de uma equipe de segurança *SOC*, porém, deixa claro que, para a melhor análise da equipe, o uso de ferramentas de *scanner* e análises em tempo real devem ser utilizadas, logo, em sua citação de *WAF*, o autor considera de extrema importância o seu uso.

Neto (2013) utiliza em seu trabalho a implementação de uma aplicação de *firewall* no processo de desenvolvimento de sistemas *web*. Em seu trabalho de análise, o autor cita o não comprometimento das empresas com a segurança dos dados, de forma que hajam muitas vulnerabilidades, trazendo como exemplos as análises de ataques nos dados de Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br) e dados do instituto *OWASP*.

Neto (2013) trata a definição de *WAF* em comparação com *firewall*, IDS e IPS. Nesta comparação, o autor classifica a posição de implementação da ferramenta *WAF*, ainda correlacionando o *WAF* com o uso de uma lista branca e uma lista negra em sua implementação. Ao longo de seu trabalho, ele classifica o uso devido para cada uma das listas, de forma a melhorar cada implementação para sua situação. Sua proposta de trabalho é a utilização da ferramenta *UniscanWAF*, que, por sua vez, é uma plataforma *WAF* que pode ser segmentada com o uso de uma lista branca. Segundo Neto (2013), é uma lista de permissões de ações dada pelos desenvolvedores da aplicação do sistema *web,* desde o seu desenvolvimento.

Durante o trabalho, o autor classifica mudanças de propostas de desenvolvimento, como testes e resultados de segurança, após a implementação da ferramenta. Em sua conclusão, o uso de um *WAF* com uma lista branca pode minimizar ainda mais as vulnerabilidades do sistema, do que apenas o uso de uma lista negra, lista predefinida. Mas cita, em sua conclusão, que a implementação da lista branca é oriunda e de responsabilidade de atualização do desenvolvedor da aplicação *web*, em seu desenvolvimento e sua atualização ao longo da utilização da aplicação.

4 METODOLOGIA

Neste capítulo são apresentados os métodos de pesquisa e desenvolvimento do trabalho, embasando os conhecimentos da revisão teórica mencionados, apresentando a abordagem, os objetivos da pesquisa, os procedimentos técnicos, as ferramentas utilizadas para o desenvolvimento, assim como as plataformas e sistemas de terceiros, cuja aplicação do trabalho é fazer a segurança.

4.1 Metodologia da pesquisa

A metodologia de pesquisa neste trabalho empregada se divide em etapas, com pesquisa bibliográfica, pesquisa exploratória, pesquisa experimental e com pesquisa aplicada. Na primeira etapa do trabalho, a pesquisa bibliográfica, segundo Lima e Mioto (2007), é caracterizada pelo ponto de vista de procedimentos técnicos, ou seja, a pesquisa consiste em coletar informações para revisão teórica, baseando-se em textos, livros, artigos e materiais científicos. Esses dados são utilizados para citações de revisões teóricas, para dar embasamento para o desenvolvimento do tema final. Nesta pesquisa, são levantados assuntos paralelos de vulnerabilidades de sistemas *web*, como os tipos de ataques, ferramentas de segurança possíveis, leis brasileiras sobre o assunto, todos para finalidade do estudo e aplicação de um *WAF*.

Após, temos a pesquisa exploratória, definida por Moretti (2017) como uma pesquisa para adquirir familiaridade com o tema, procurando saber o funcionamento do assunto, em uma pesquisa flexível e não estruturada. Essa que, por sua vez, é uma pesquisa de exploração sobre o assunto final, onde é realizada a pesquisa sobre

a ferramenta a ser utilizada para solução do problema, assim como aplicar a solução em detalhes. Nesta etapa, é realizada a pesquisa do *ModSecurity* em conjunto com regras da *OWASP*, para melhor aplicação da ferramenta e como aplicá-la.

A outra etapa da pesquisa é empregada em uma pesquisa experimental, definida por Moretti (2017) como uma pesquisa de continuidade às pesquisas exploratórias, onde o estudo realizado anteriormente agora será aplicado em fase de testes, até atingir um resultado satisfatório. Nesta etapa da pesquisa, é dada a aplicação em fase de testes da ferramenta *WAF*, como experimental, ajustando os detalhes com testes efetivos para analisar se as regras aplicadas na ferramenta são adequadas e atendem os requisitos de segurança, caso contrário, nova etapa experimental se inicia, modificando as regras até que seja adequada.

Na última etapa da metodologia temos a pesquisa aplicada, pesquisa que, segundo Tumelero (2019), visa produzir um conhecimento em um objetivo de problema real, ajudando a melhorar a situação do fenômeno. Pesquisa essa, que é aplicada em um complemento ou aprofundamento de um assunto previamente estudado, com uma proposta de apresentar alternativas de soluções para melhorar o problema do assunto, transformando determinadas ferramentas em soluções do estudo.

A solução que pode ser aplicada com um desenvolvimento experimental, gerando conhecimentos para aplicação prática do estudo, realizando objetivos de conhecimentos e melhorias em curto e médio prazo, investigando e adquirindo novos conhecimentos, alcançando novos métodos ou maneiras de alcançar um certo objetivo e aplicando o conhecimento do estudo, visando utilidade econômica e social.

Nesta pesquisa, aplicamos a ferramenta *WAF* para assim proteger de forma efetiva o sistema da empresa AWTI Sistema de Informação, essa que, por sua vez, é uma empresa especializada em desenvolvimento de *sites*, sistemas empresariais, consultoria em Tecnologia de Informação que não provém de nenhuma segurança para suas aplicações *web*.

4.2 Propósito da implementação

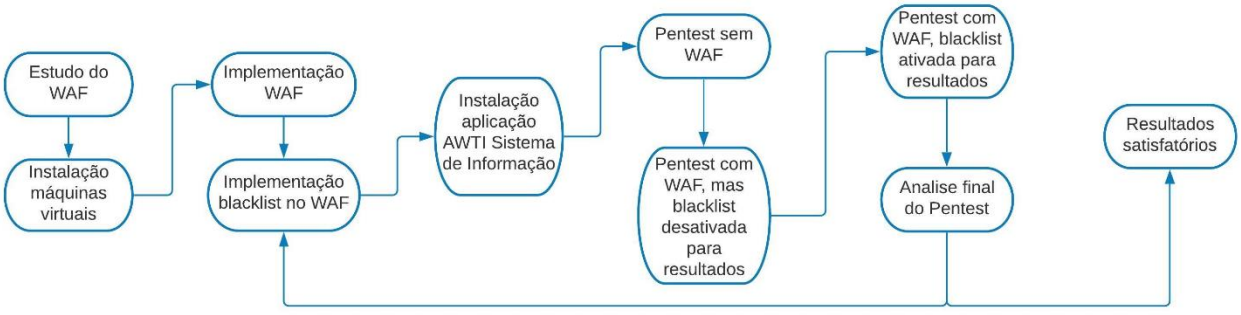
O presente estudo possui dois propósitos, o primeiro consiste na implementação de um *WAF* em uma máquina virtual, juntamente com aplicação *web* de terceiro, de forma que o *WAF* assegure os dados na aplicação e minimize as vulnerabilidades e ataques. A solução de *WAF* implementada neste estudo será dada pela distribuição *ModSecurity*, com utilização apenas no método de *blacklist*. O segundo objetivo do estudo é levantar análises possíveis para avaliar se o *WAF* implementado realmente minimiza os riscos de vulnerabilidades da aplicação *web*, utilizando ferramentas de *Pentest* que avaliam as vulnerabilidade e ataques à aplicação *web*, desta forma, obtendo resultados para análise se o *WAF* atingiu as expectativas de segurança em vulnerabilidades e ataques propostas.

4.3 Fluxograma de implementação

Para compreensão e implementação do *WAF* e as análises a serem feitas, a Figura 7 demonstra o fluxograma para entendimento do estudo da pesquisa de implementação.

Desta forma, a revisão teórica se aplica inicialmente para um estudo e embasamento do objetivo da implementação do *WAF* e a necessidade de atender a normas, leis e segurança da aplicação *web* da empresa AWTI Sistemas de Informação.

Figura 7 – Fluxograma de implementação



Fonte: Do autor (2020).

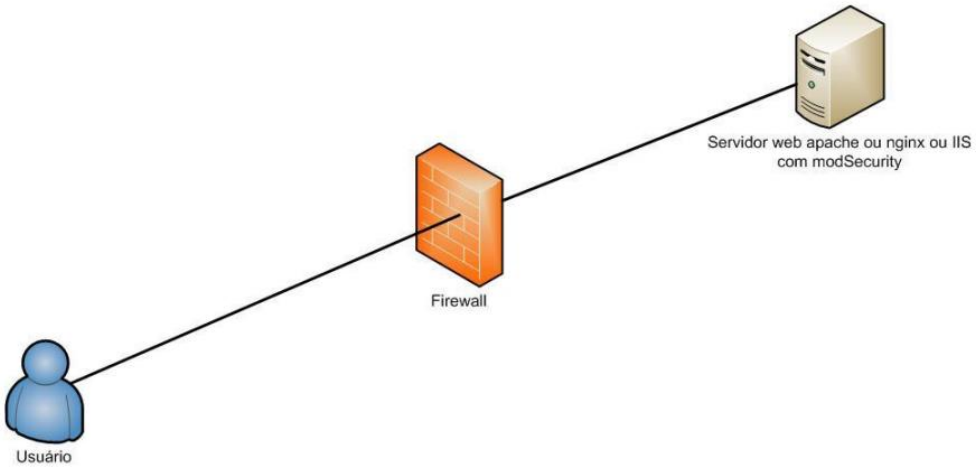
No fluxograma da Figura 7, podemos observar o início do estudo, em um modo de aplicação do *WAF* seguido pela instalação das máquinas virtuais, necessárias para implementação das ferramentas em laboratório controlado, para assim determinar melhor análise dos resultados. O primeiro passo após é a implementação do *WAF ModSecurity*, seguido pela implementação da *blacklist OWASP ModSecurity Core Rule Set* (CRS), seguido pela instalação da aplicação *web* da empresa AWTI Sistemas de Informação.

Nesta fase, se deu o início dos testes de análises com a ferramenta *Skipfish e OWASP ZAP*, inicialmente sem a utilização do *WAF*, seguido pelos testes com a implementação do *WAF*, mas sem o uso da *blacklist*, para determinação de performance da aplicação *web*, seguido pela análise de *Pentest* com *WAF* e *blacklist* ativos, para relatar a segurança do uso da ferramenta, para então determinar se a aplicação no modo aplicado possui resultados satisfatórios ou necessita novas orientações de implementação da *blacklist*.

4.4 Implantação em modo incorporado

Este foi o modelo utilizado nesta aplicação, dessa forma, o *ModSecurity* trabalha incorporado ao servidor *Web*, utilizando os mesmos recursos do servidor para a aplicação *web* e para o *WAF*. Esta é uma ótima escolha para empresas onde a arquitetura de rede já está definida, ainda lembrando que seja necessário o baixo custo de implementação, por isso o uso de máquina virtual e o compartilhamento dos recursos de *hardware*, conforme Figura 8 (RISTIC, 2010).

Figura 8 – Funcionamento *WAF* modo incorporado



Fonte: Stein (2013, p. 17).

Na Figura 8 temos o usuário acessando uma aplicação *web*, onde seu tráfego passa primeiramente por um *firewall* e após é direcionado para o servidor *Apache*, *Nginx* ou IIS com a ferramenta *WAF ModSecurity*, compartilhadas no mesmo *hardware*.

4.5 Máquina virtual

Para aplicação do *WAF*, a ferramenta será instalada em uma máquina virtual, inicialmente levantada em um computador com *Windows* 10, versão 19041, utilizando o sistema de virtualização *Hyper-V*.

A máquina virtual *Hyper-V* será utilizada com um *hardware* de 2 processadores virtuais, 4 *gigabytes* de memória *RAM*, disco de armazenamento SSD de 30 *gigabytes*, e uma placa de rede. O *hardware* compartilhado para a instalação do *Hyper-V* será um processador *Intel* (R) *Core* (TM) i5-760, 16 *gigabytes* de memória *RAM*, disco SSD 480 *gigabytes* e uma placa de rede *gigalan* (10/100/1000 Mb/s). Os testes serão realizados em uma rede interna com equipamentos via cabo de rede e *switch gigalan* para uma maior performance nos ataques.

4.6 *Ubuntu server*

A plataforma de sistema operacional utilizada neste trabalho será o *Linux Ubuntu Server* 20.04 LTS, versão atual da distribuição *Linux Ubuntu*. Apenas o modo *Kernel* é necessário, desta forma, a distribuição *Linux Ubuntu* utilizada não necessita um *hardware* com grande performance. Podendo ser rodada sem muitas exigências em uma máquina virtual com *hardware* descrito no capítulo anterior.

4.7 Instalação *ModSecurity* no *Ubuntu server*

O *ModSecurity* instalado na distribuição *Linux Ubuntu Server* 20.04 LTS, é implementada como uma biblioteca de aplicação, desta forma, o *ModSecurity* é instalado com o utilitário de instalação de bibliotecas do *Linux* (APT-*GET*), em modo administrador (*root*).

4.8 *Blacklist ModSecurity* e *OWASP*

Para implementação de uma *blacklist*, o *ModSecurity* necessita uma lista com regras para a *blacklist*, a qual não vem pré-configurada. Nesta aplicação, a lista implementada no *ModSecurity* será a lista em conjunto criada pela *ModSecurity* em união com o instituto *OWASP*, projeto *OWASP ModSecurity Core Rule Set* (CRS). Será utilizada a lista com a versão 3.2.0, com todas as regras da lista no modo padrão, sem serem alteradas.

4.9 Aplicação *web*

A aplicação *web* da empresa AWTI Sistemas de Informação é um sistema com código fonte, em linguagem PHP, utilizado como base para novas aplicações o qual

possui uma tela de *login* com consulta ao banco de dados, desta forma, o sistema valida quais usuários podem acessar a aplicação. Dentro do sistema, possui um cadastro simples de cliente, cadastro utilizado para controle interno da empresa.

A aplicação será implementada na mesma máquina virtual do *ModSecurity*. O sistema operacional utilizado será da distribuição *Linux Ubuntu Server* 20.04 LTS, juntamente com o serviço de servidor HTTP *Apache* versão 2.2.4, plataforma de linguagem *script* PHP 5.6, os dados são armazenados no banco de dados da *PostgreSQL* em sua versão 9.4.

4.10 Ferramenta de *Pentest*

São duas ferramentas escolhidas para *Pentest*. A primeira ferramenta de *scanner* para *Pentest*, utilizada para avaliação das vulnerabilidades e ataques ao site *web*, será a aplicação de *Pentest Skipfish*. Utilizando as vulnerabilidades e ataques à aplicação *web*, com base nos maiores ataques, conforme a lista da OWASP (2017).

A ferramenta *Skipfish* foi escolhida por ter obtido maiores resultados em análises no trabalho acadêmico de Costa (2017), o qual relaciona ataques a sites *web* governamentais. O autor constou resultados de vulnerabilidades, com o *Skipfish*, muito acima de outras ferramentas apresentadas.

Segundo Costa (2017), as ferramentas foram analisadas de acordo com as funcionalidades:

- Relatório: detalhamento de cada vulnerabilidade, indicando o ponto de acontecimento, com persistência para o dado de relatório;
- Usabilidade: facilidade de utilizar a ferramenta;
- Eficácia: quantidade de vulnerabilidades e ataques detectadas pela ferramenta.

Com essas funcionalidades, Costa (2017) atribui um quadro de resultados apresentados, classificando as ferramentas com nota de 1 a 3 para cada funcionalidade, onde 1 significa que a ferramenta não atendeu às exigências, 2 atendeu e 3 possui exigências acima do necessário, conforme Quadro 1.

Quadro 1 – Avaliação das ferramentas *Pentest*

Estado	Relatório	Usabilidade	Eficácia
Skipfish	3	2	3
Uniscan	2	2	3
Vega	1	3	3
W3AF	1	2	3

Fonte: Costa (2017, p. 27).

Os resultados dos testes de Costa (2017) apresentam a ferramenta de *Pentest Skipfish* como uma das melhores opções de análises para aplicações *web*, onde possui o melhor resultado de relatório entre as ferramentas apresentadas, assim como atende os requisitos de facilidade de usabilidade e possui exigências acima do necessário em sua eficácia de encontrar vulnerabilidades.

A segunda ferramenta de *Pentest* utilizada para avaliação das vulnerabilidades e ataques ao site *web* será a aplicação de *Pentest OWASP Zep Attack Proxy (ZAP).* Ferramenta indicada pela instituição, em conjunto com as vulnerabilidades já descritas anteriormente pela própria instituição. A ferramenta *ZAP* utiliza um *script* de ataque pré-definido, seguindo uma lista de ataques indicada pela instituição, fazendo todos os testes de vulnerabilidade, juntamente com os registros de cada ataque em forma HTML, para avaliação posterior.

As duas ferramentas, *Skipfish* e *ZAP, são* implementadas em cima da distribuição *Linux Kali*, na versão 2017.2, a qual será utilizada para execução das análises. Distribuição essa que será implementada em outra máquina virtual do *Hyper-V*, com as mesmas configurações da distribuição *Linux Ubuntu Server* 20.04 LTS, apresentada anteriormente.

4.11 Testes de vulnerabilidade e ataques com *Pentest*

Os testes de avaliação da necessidade do uso de uma ferramenta *WAF* foi baseado em duas baterias de testes. A primeira com o *WAF* desligado, atacando a aplicação *web* diretamente com a ferramenta de *Pentest Skipfish* e a ferramenta de *Pentest OWASP ZAP*, obtendo resultados apenas na ferramenta de *Pentest*. Ainda

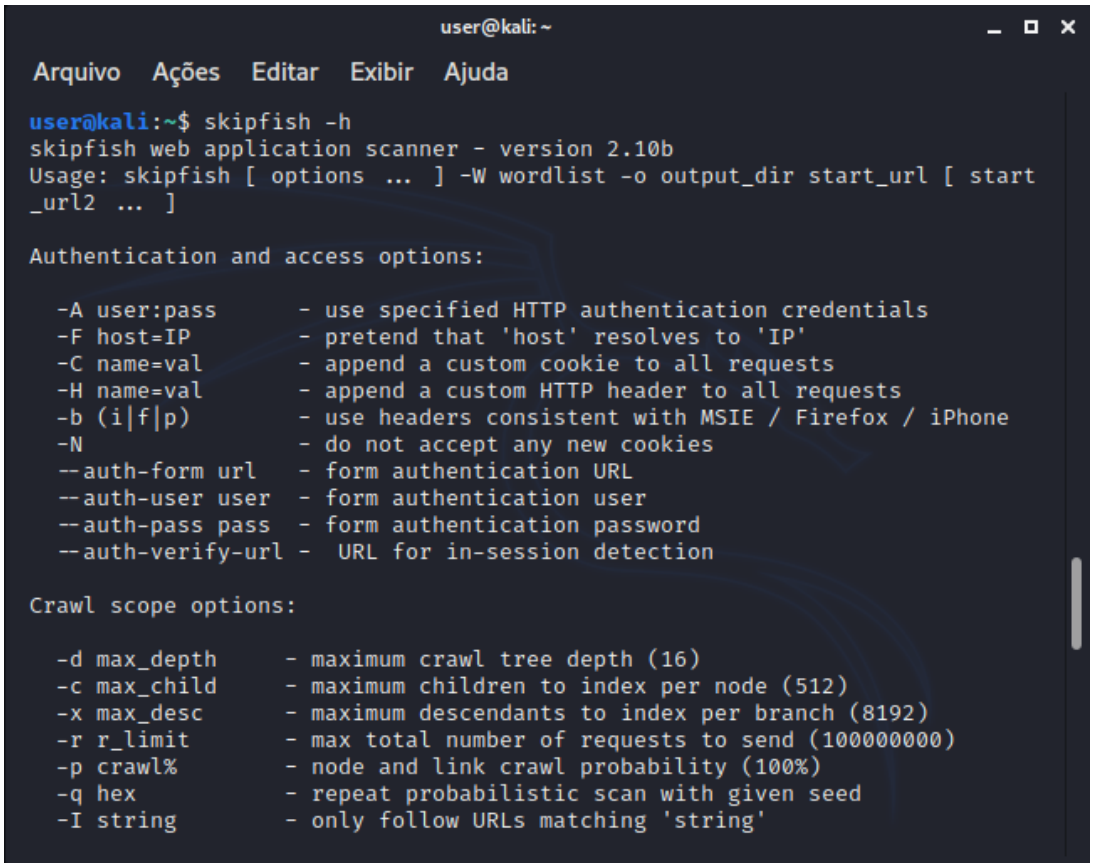
nesta bateria de testes, será analisado o uso da ferramenta *WAF* para análise de performance do *Apache* ao ser atacado pelos *Pentest*, mas nesta etapa não será utilizado as regras de *blacklist* do *ModSecurity*.

Na segunda bateria de testes será adicionada a *blacklist OWASP ModSecurity Core Rule Set* (CRS), conforme determinado em capítulos anteriores, a *blacklist* será utilizada sem modificações. Os testes nesta bateria serão os mesmos da primeira bateria de testes, para ter a comparação dos eventuais testes realizados pelo *Pentest Skipfish* e *Pentest OWASP ZAP*.

Os testes realizados pelo *Skipfish* são baseados em um dicionário, mapeando um *site* interativo para a aplicação de destino, com testes de rastreamento recursivo, baseado nas análises de um dicionário. Os resultados do *Pentest* são anotados em um relatório final pela própria ferramenta *Skipfish*, o relatório pode servir como base para avaliações de segurança de aplicativos *web* (SKIPFISH, 2018).

Segundo a Skipfish (2018), para executar toda a lista de testes de vulnerabilidades e ataques da lista de dicionário da ferramenta *Skipfish*, é necessário que a ferramenta seja instalada no *Kali Linux*, conforme mencionado em capítulos anteriores, e seja executado em modo *root* pelo *Kernel* do *Linux*. O primeiro comando a ser executado deve ser a chamada da aplicação *Skipfish* com o parâmetro -h (*skipfish* -h), demonstrando toda a lista de parâmetros da ferramenta, conforme demonstrado na Figura 9.

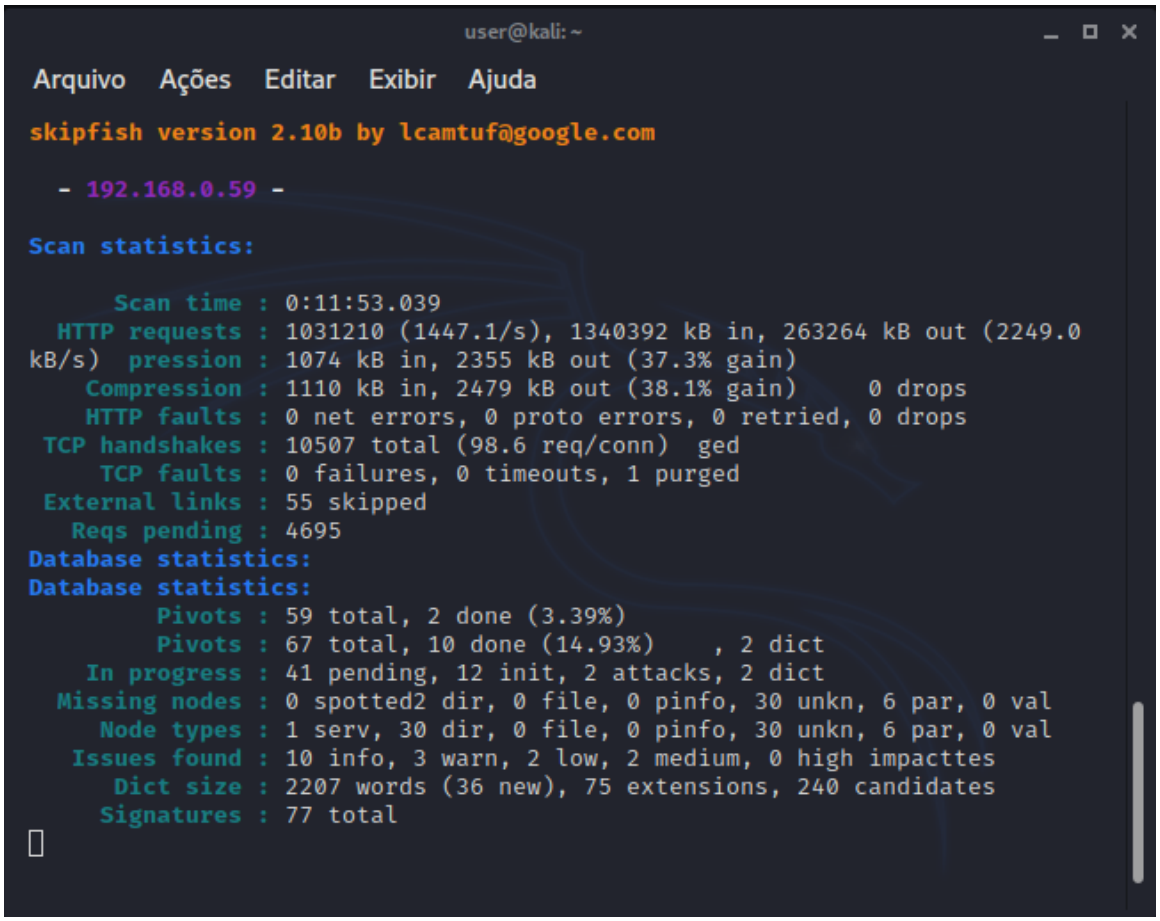
Figura 9 – Executando *Skipfish*



Fonte: Do autor (2020).

O próximo passo do *Pentest* com *Skipfish*, é a execução dos testes de vulnerabilidades e ataques em cima da aplicação de destino. Este passo pode ser iniciado com o comando *skipfish* -o “caminho destino registros”, -S “caminho destino dicionário”, “caminho de destino do sistema”. Onde -o serve para informar o caminho destino dos registros dos testes, -S serve para informar a biblioteca de ataque (*Wordlist*). Por final, é dado o parâmetro do endereço da aplicação *web*, dada normalmente pelo *link* HTTP de uma aplicação do *Apache*. Desta forma, temos o exemplo de execução conforme a Figura 10, com um simples teste de ataque a um sistema com destino de IP 192.168.0.59.

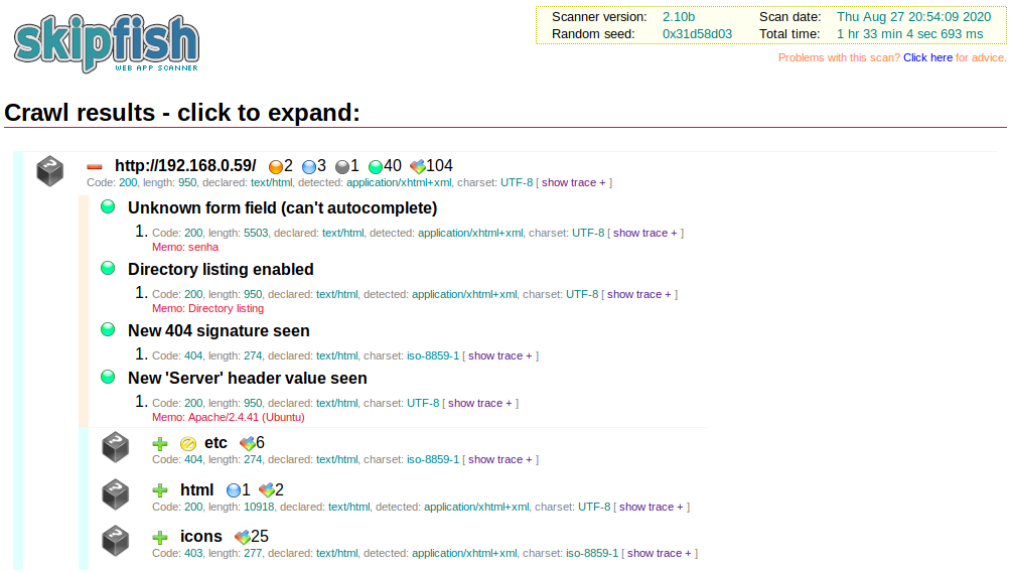
Figura 10 – Executando os testes com *Skipfish*



Fonte: Do autor (2020).

Assim que os resultados forem exibidos no modo *Kernel*, os mesmos foram disponíveis em modelo HTML para análises mais precisas, a página HTML será demonstrada com vários parâmetros de análises, podendo ser mais detalhado em cada teste de vulnerabilidade, caso seja necessário, conforme Figura 11.

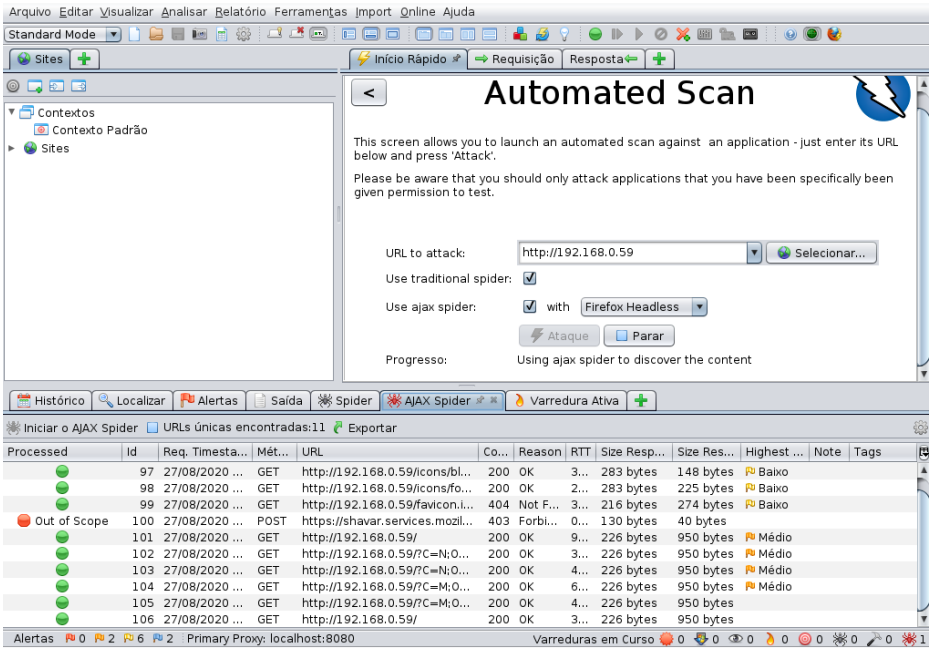
Figura 11 – Exibindo os testes com *Skipfish*



Fonte: Do autor (2020).

A segunda ferramenta de *Pentest* utilizada é do instituto *OWASP*, ferramenta cuja proposta é realizar ataques de todos os tipos, seja em *firewall*, *WAF*, *proxy* e outros serviços de segurança. A ferramenta *OWASP ZAP* na versão 2.9.0 é a mais recente disponível, seu uso é via aplicação visual, basta apenas indicar o caminho da aplicação. Na Figura 12, temos um exemplo de ataque à mesma aplicação anterior. Após, os resultados são exibidos na mesma janela.

Figura 12 – Executando os testes com *OWASP ZAP*



Fonte: Do autor (2020).

4.12 Análise dos resultados

Aplicando o *Pentest Skipfish* e a ferramenta *WAF* do *ModSecurity*, obtendo os resultados com e sem a segurança do *WAF* a uma aplicação *web*, podemos analisar os resultados e compará-los entre si, com o objetivo de demonstrar para empresa AWTI Sistemas de Informação que o uso de uma ferramenta *WAF* para proteger seus sistemas *web* é de uso necessário, apresentando resultados satisfatórios e atendendo normas e leis de segurança para aplicações nacionais existentes e que entraram em vigor, citadas em capítulos anteriores.

A análise dos testes influencia em novos testes, com novos parâmetros, aplicando uma nova *blacklist* ao *ModSecurity*, caso a *blacklist OWASP ModSecurity Core Rule Set* (CRS) não atenda resultados satisfatórios, ou ainda podendo trocar a ferramenta *WAF ModSecurity* por outra ferramenta *WAF*, para novas avaliações e possíveis comparações entre ferramentas *WAF*, para assim, atender o objetivo do trabalho de proteger a aplicação *web* e atender às normas e leis de segurança nacionais.

4.13 Instalação de máquinas virtuais e Sistemas Operacionais

O primeiro passo dos testes foi criar duas máquinas virtuais, uma para aplicação *web* da AWTI Sistemas de Informação e outra do *Kali Linux* para realizar os ataques, conforme relacionado nos capítulos anteriores.

Ambas as máquinas foram gerenciadas pelo *Microsoft Windows* 10, através da plataforma *Hyper-V*, com *hardware* disponível conforme capítulos anteriores.

O segundo passo foi a inicialização e instalação dos Sistemas Operacionais, uma com *Ubuntu Server* 20.04 LTS e outra com *Kali Linux* 2020.3, onde o *Ubuntu* apresenta apenas interface de modo *Kernel* (sistema de baixo nível do *Linux*), conforme Figura 13, enquanto a instalação do *Kali Linux* apresenta a interface de modo usuário (gráfico), e configurações padrões de instalação que vêm sem modificações em suas bases.

Figura 13 – *Ubuntu Server*

Fonte: Autor (2020).

Na Figura 13 temos a janela inicial do *Ubuntu Server*, mesma apresentada nas máquinas onde foram instaladas as aplicações de clientes da empresa AWTI Sistemas de Informação. A imagem apresenta o comando “cat /etc/*-release” para demonstrar a versão do *Ubuntu Server* que fora instalada.

E no *Kali Linux* temos as ferramentas atualizadas para fazer os ataques de *Pentest* à aplicação *web*. O *Kali Linux* é uma distribuição *Linux* que já consta pré-instalado as últimas versões de *softwares* para ataques a sistemas, neste caso, os testes foram feitos pelo *software Skipfish* e *OWASP ZAP*. O próximo passo foi instalar e configurar o *Apache* no *Ubuntu Server*.

4.14 Instalação do servidor *Apache*

Essa foi a ferramenta base principal, foi o servidor da aplicação *web*, onde também foi a base do *ModSecurity* e do complemento *Mod Evasive.* A versão do *Apache* instalada foi a mesma versão aplicada na empresa AWTI Sistema de Informações. A instalação do *Apache* é dada pelo gerenciador de pacotes do *Linux* (APT-*GET*), desta forma, sua instalação foi simples, apenas com o comando *sudo apt install apache2*. Comando *sudo* para permissão de *root*, *apt* para chamada do gerenciador de pacote, *install* para comando de instalar, *apache*2 o pacote do *apache* na última versão disponível (CANONICAL, 2020).

Para configurar o acesso e permitir conexões externas para aplicação de teste no *Apache*, foi necessário configurar o arquivo do *Apache* e modificar o caminho

/var/www/html para /var/www/, modificação feita acessando o arquivo /etc/apache2/sites-enabled/000-default.conf pelo comando *sudo nano* /etc/apache2/apache2.conf, *sudo* para permissão *root*, *nano* é a chamada do editor de texto via *Kernel*, e o caminho completo do arquivo de configuração do *Apache* (CANONICAL, 2020).

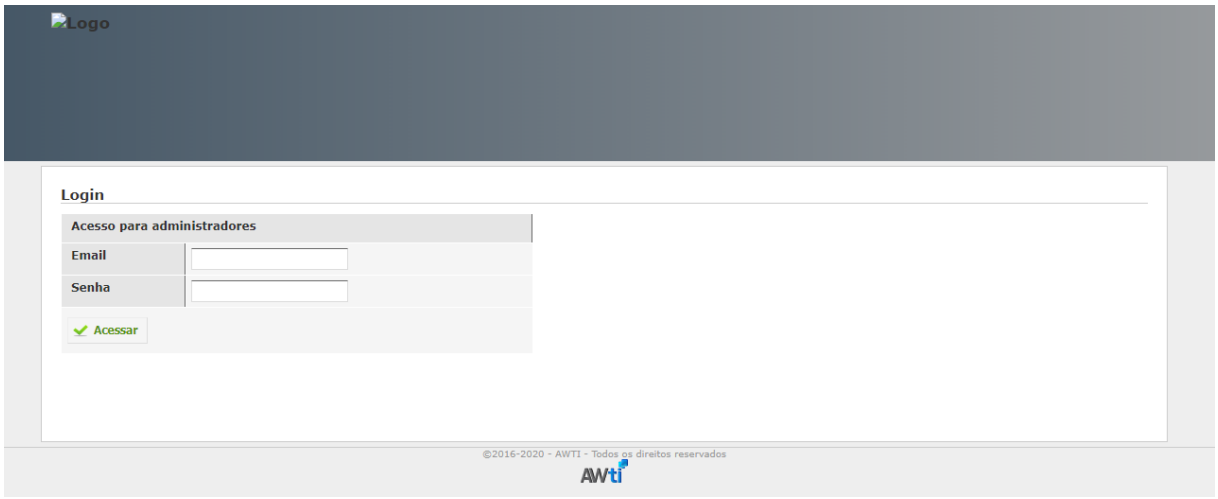
4.15 Instalação e configuração PHP 5.6 e *PostgresSQL* 9.4

Para implementar o sistema da empresa AWTI Sistemas de Informação, foi necessário a instalação dos módulos do *Apache* para linguagem PHP, estes também instalados pelo gerenciador de pacotes do *Linux*, com o comando *sudo apt install php5.6*, *sudo* para permissão *root*, *apt* chamada do gerenciador de pacote, *install* para comando de instalar e *php5.6* o nome do pacote, no qual vem incluso os pacotes adicionais necessários para rodar arquivos de programação com extensão *php* no servidor *Apache*. Não foi utilizada a última versão da linguagem, devido à base de programação do sistema da empresa AWTI Sistemas de Informação ser desenvolvida na versão 5.6. Após a instalação, apenas tivemos que habilitar a linguagem no arquivo de configuração do *Apache*, adicionando a linha que inclui a linguagem PHP (PHP.COM.BR, 2020).

Já para instalação do banco de dados *PostgreSQL* 9.4, foi necessário incrementar à lista de endereços do gerenciador de pacotes do *Linux.* E, após, instalar apenas com o comando do gerenciador de pacotes do *Linux* (POSTGRESQL, 2020).

Com a instalação de todos os pacotes necessários, podemos implementar o sistema de testes fornecido pela empresa AWTI Sistema de Informações, juntamente com banco de dados com informações e páginas de *login* e cadastro de dados, conforme a Figura 14.

Figura 14 – Sistema de teste



Fonte: Autor (2020).

A Figura 14 conta com uma tela de *login*, essa que não possui tratamento de vulnerabilidade contra tentativas de *login*, onde os campos fazem consulta ao banco de dados *PostgreSQL*, deixando uma grande vulnerabilidade para ataques. As demais pastas de arquivos do sistema podem ser acessadas sem segurança, permitindo a visualização dos arquivos sem restrições de segurança, deixando a aplicação e o servidor vulneráveis.

Com estas informações e o sistema de teste implementado, realizamos os testes de vulnerabilidades, e seus resultados são apresentados e discutidos no próximo capítulo.

5 RESULTADOS E DISCUSSÃO

Para fins de avaliação do projeto, foram realizados testes práticos conforme mencionado nos capítulos anteriores. Em laboratório foi efetivada a implementação do *WAF* em máquina virtual, e realizado testes de ataques até o momento em que a finalidade de segurança chegou ao resultado satisfatório. Após foi repassado os testes e análises de estudo para a empresa AWTI Sistemas de Informação, juntamente com os resultados do antes e depois da aplicação de segurança do *WAF*.

As etapas seguintes foram do fluxograma no capítulo 4.3, onde primeiramente foi instalado o servidor *Apache* junto com o sistema *web* da empresa AWTI Sistemas de Informação, após, foram realizados os eventuais testes de vulnerabilidades com a ferramenta *Skipfish* e *OWASP ZAP*. Seguido dos resultados desses testes, foi implementado a ferramenta *WAF ModSecurity*. Então, foram realizados novamente os mesmos testes, sendo executados da mesma maneira, onde foram obtidas algumas implementações de regras CRS não satisfatórias, realizando novas configurações de regras CRS, até que os resultados obtidos foram satisfatórios, e, após, foi realizada a comparação dos resultados.

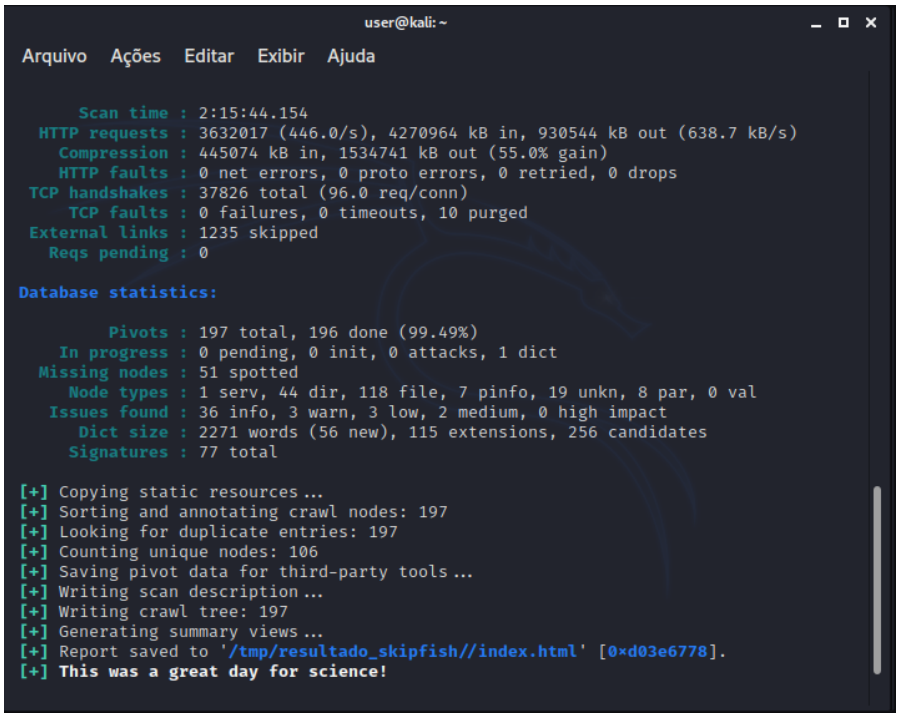
5.1 *Pentest Skipfish* sem *WAF*

O primeiro teste de vulnerabilidade que foi executado, com o *software* de *Pentest Skipfish* e a biblioteca de teste em modo moderado, gerou alguns resultados de vulnerabilidade de segurança, mostrando que o sistema de teste da empresa AWTI Sistemas de Informação não possui uma boa segurança. Para melhor testar as

seguranças e vulnerabilidades do teste, foi tomado como padrão a utilização da biblioteca de testes completa do *Skipfish*, padronizando os ataques e testes para que assim, no final de todas as baterias de testes, tivesse uma comparação com os mesmos parâmetros.

Os testes realizados foram efetuados com uma bateria de 10 vezes consecutivas, pois houve uma leve diferença nos resultados das informações obtidas, mas não foram observado mudanças no resultado das criticidades. Com essa atribuição, o teste do *Skipfish* sem segurança *WAF* implementada foi dado pelo comando *sudo skipfish -o* /tmp/resultado_skipfish/semwaf/ -S /usr/share/skipfish/dictionaries/complete.wl http://192.168.0.59/sistemateste, onde *sudo* é o comando de permissão para modo *root*, *skipfish* a chamada do *software* de *Pentest*, *-o* parâmetro para definir local para salvar resultados, após o caminho onde foram salvos os resultados do *Pentest*, *-S* para definir a biblioteca manualmente, após o caminho da biblioteca, no caso, aplicado a biblioteca completa, utilizada como padrão definido para todos os testes e baterias realizadas, e por fim, o caminho completo do sistema *web*. Desta forma, tivemos um teste realizado conforme a Figura 15.

Figura 15 – *Pentest Skipfish* sem *WAF*



Fonte: Autor (2020).

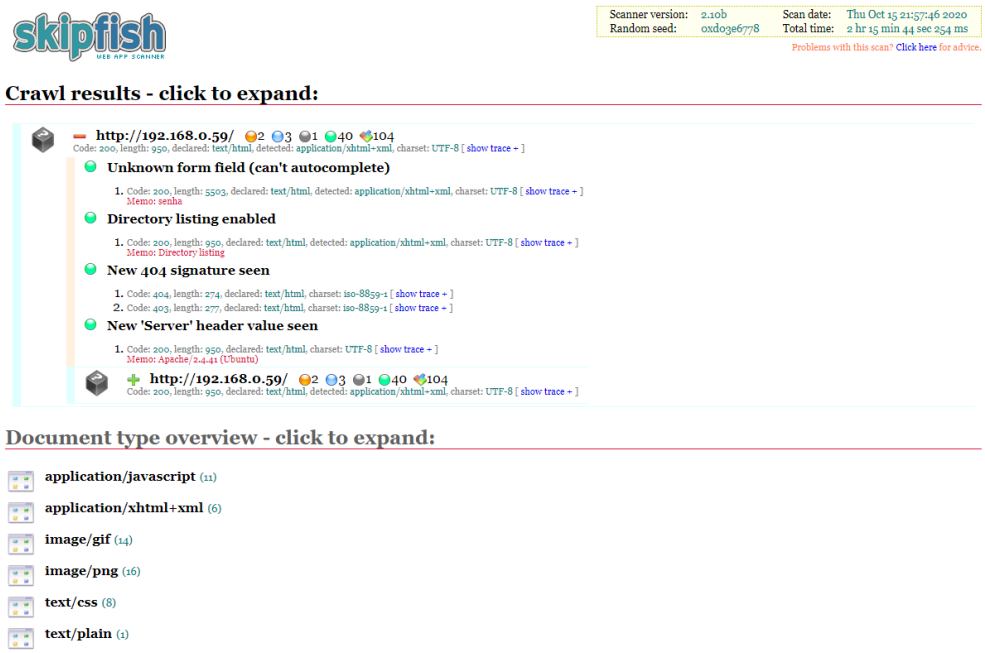
O teste realizado na Figura 15 apresenta o resultado de uma das 10 vezes da bateria de testes, todos ocorreram pelo parâmetro padrão, sobretudo, dois testes tiveram uma demora a mais nos resultados, passando de três horas de ataques em cada teste. As demais 8 vezes da bateria realizada ficaram em um tempo próximo de duas horas de duração para cada teste realizado.

Seguindo os resultados da Figura 15, podemos observar que foram realizadas 3.632.017 requisições de ataques no protocolo HTTP, com nenhum erro gerado, nem mesmo um pacote de dados perdido.

Nas estatísticas da Figura 15, observamos que foram realizados 197 tipos de ataques diferentes, resultando nos mais de três milhões de ataques realizados. Dentre eles, temos os resultados do *Apache* com 1 servidor, 44 pastas de arquivos, 118 arquivos, 7 informações de diretórios, 19 resultados não definidos e logo abaixo os resultados de ataques definidos por nível de vulnerabilidade, com nenhum impacto de criticidade alto, 2 com criticidade média, 3 baixos, 3 alarmes e 36 informações.

Após o fim de cada bateria de testes, os resultados foram analisados individualmente, investigando cada vulnerabilidade apresentada pelo *Skipfish*, reportada no arquivo HTML criado pela aplicação de *Pentest*, encontrada no destino passado no comando inicial.

Figura 16 – Resultados *Pentest Skipfish* sem *WAF*



Fonte: Autor (2020).

Na Figura 16, podemos observar inicialmente os mesmos resultados de vulnerabilidade destacados na Figura 15, conforme o nível de criticidade da vulnerabilidade. Sobretudo, os resultados do *Skipfish* unem as informações obtidas para melhor visualização. Dentre os resultados do cabeçalho temos apenas alguns alertas de informações disponíveis do servidor *Apache*, nada gerado pelo sistema da empresa AWTI Sistemas de Informação, ainda na guia separada algumas observações com quantidade de arquivos lidas nos diretórios e as respectivas extensões. Seguindo os resultados, observamos cada ataque de vulnerabilidade com o resultado, conforme na Figura 17.

Figura 17 – Resultados detalhados *Pentest Skipfish* sem *WAF*

External content embedded on a page (higher risk) (2)
1. http://192.168.0.59/sistemateste/index.php/login/ [show trace +]
   Memo: https://kit.fontawesome.com/46da2bf9a7.js
2. http://192.168.0.59/sistemateste/login/ [show trace +]
   Memo: https://kit.fontawesome.com/46da2bf9a7.js

Signature match detected (1)
1. http://192.168.0.59/html/php.php [show trace +]
   Memo: phpinfo() page (sig: 11001)

External content embedded on a page (lower risk) (2)
1. http://192.168.0.59/sistemateste/index.php/login/ [show trace +]
   Memo: http://www.awti.com.br/imagens/logo_credito_50_25.png
2. http://192.168.0.59/sistemateste/login/ [show trace +]
   Memo: http://www.awti.com.br/imagens/logo_credito_50_25.png

Parent behavior checks failed (no brute force) (1)
1. http://192.168.0.59/sistemateste/sfi9876/css/ [show trace +]

Numerical filename - consider enumerating (5)
1. http://192.168.0.59/sistemateste/0/ [show trace +]
2. http://192.168.0.59/sistemateste/css/ui-lightness/jquery-ui-1.8.19.custom.css [show trace +]
3. http://192.168.0.59/sistemateste/js/jquery-1.7.2.min.js [show trace +]
4. http://192.168.0.59/sistemateste/js/jquery-ui-1.8.19.custom.min.js [show trace +]
5. http://192.168.0.59/sistemateste/js/jquery.maskedinput-1.3.min.js [show trace +]

Incorrect or missing charset (low risk) (20)
1. http://192.168.0.59/sistemateste/css/ui-lightness/jquery-ui-1.8.19.custom.css [show trace +]
2. http://192.168.0.59/sistemateste/css/ajax-loader.css [show trace +]
3. http://192.168.0.59/sistemateste/css/buttons.css [show trace +]
4. http://192.168.0.59/sistemateste/css/formulario.css [show trace +]
5. http://192.168.0.59/sistemateste/css/jquery.cleditor.css [show trace +]
6. http://192.168.0.59/sistemateste/css/style.css [show trace +]
7. http://192.168.0.59/sistemateste/css/style_print.css [show trace +]
8. http://192.168.0.59/sistemateste/css/table.css [show trace +]
9. http://192.168.0.59/sistemateste/js/ajax-loader.js [show trace +]
10. http://192.168.0.59/sistemateste/js/commands.js [show trace +]
11. http://192.168.0.59/sistemateste/js/jquery-1.7.2.min.js [show trace +]
12. http://192.168.0.59/sistemateste/js/jquery-ui-1.8.19.custom.min.js [show trace +]
13. http://192.168.0.59/sistemateste/js/jquery.cleditor.min.js [show trace +]
14. http://192.168.0.59/sistemateste/js/jquery.columnfilters.js [show trace +]
15. http://192.168.0.59/sistemateste/js/jquery.filtertable.js [show trace +]
16. http://192.168.0.59/sistemateste/js/jquery.maskedinput-1.3.min.js [show trace +]
17. http://192.168.0.59/sistemateste/js/jquery.maskMoney.js [show trace +]
18. http://192.168.0.59/sistemateste/js/phpjs.js [show trace +]
19. http://192.168.0.59/sistemateste/js/table_csv_converter.js [show trace +]
20. http://192.168.0.59/sistemateste/plupload-3.0-beta1/js/plupload.min.js [show trace +]

Incorrect or missing MIME type (low risk) (1)
1. http://192.168.0.59/sistemateste/favicon.ico [show trace +]
   Memo: image/png

Password entry form - consider brute-force (2)

Fonte: Autor (2020).

Com estes resultados da Figura 17, observamos que apesar dos riscos encontrados pelo *Skipfish* não serem de criticidade alta, os resultados de nível médio foram definidos como altos dentro do parâmetro médio, contando com vulnerabilidades como possível ataque a *SQL Injection*, esta que é uma das maiores e mais perigosas vulnerabilidades conforme a lista *OWASP*, apresentada no capítulo

3.1.3. Para tal compreensão da possibilidade, o *Skipfish* detalha o requisito do ataque, analisado na Figura 18.

Os demais resultados da Figura 17, em sua maioria, são inofensivas as informações da aplicação *web*, com apenas resultados de caminhos externos dados por informações das aplicações da linguagem *JavaScript,* contida em algumas páginas da aplicação. Informações essas, dadas por busca de valores externos pela aplicação. Alguns arquivos de *Layout* de página também contam como vulnerabilidades, porém de criticidade baixa, por não terem nenhum tratamento de segurança e terem as informações disponibilizadas.

Figura 18 – Resultados detalhados *Pentest Skipfish* sem *WAF* com maior risco

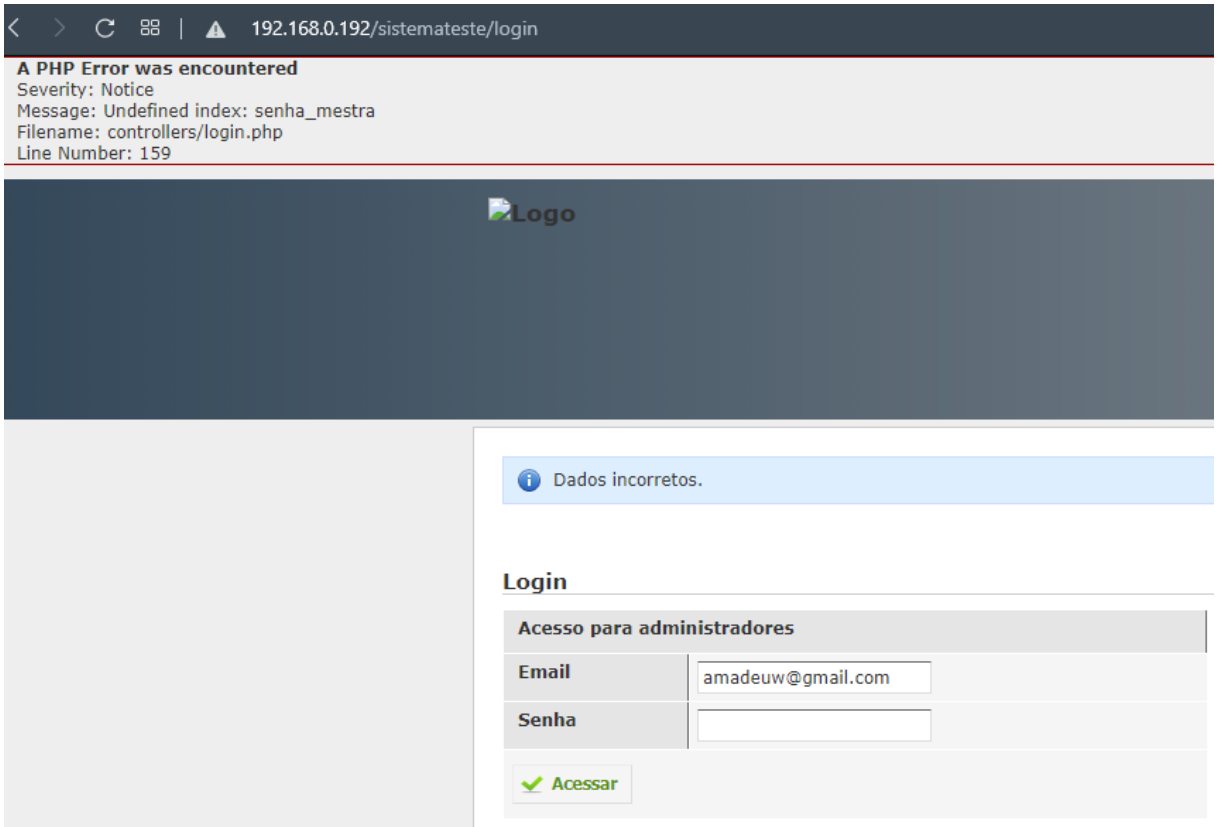
```
=== REQUEST ===

GET /sistemateste/index.php/login/ HTTP/1.1
Host: 192.168.0.59
Accept-Encoding: gzip
Connection: keep-alive
User-Agent: Mozilla/5.0 SF/2.10b
Range: bytes=0-399999
Referer: http://192.168.0.59/
Accept: skip/fish;
Cookie: PHPSESSID=bgvkbngdfkg94o8rr2ggn4f084


=== RESPONSE ===

HTTP/1.1 200 Partial Content
Date: Thu, 15 Oct 2020 22:53:31 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Range: bytes 0-1574/1575
Content-Length: 1575
Keep-Alive: timeout=5, max=48
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Teste</title>

    <!-- Meta tags -->
    <meta charset="utf-8">
    <meta http-equiv="content-language" content="pt">
    <meta name="keywords" content="Teste">
    <meta name="description" content="">
    <meta name="author" content="AWti Sistemas de InformaÃ§Ã£o">
    <meta name="robots" content="index,follow">
    <!-- Meta tags -->

    <link rel="shortcut icon" href="http://192.168.0.59/sistemateste/favicon.ico" />
    <link rel="icon" href="http://192.168.0.59/sistemateste/favicon.ico" type="image/x-icon" />

    <link rel="stylesheet" type="text/css" href="http://192.168.0.59/sistemateste/css/style.css" />
    <link rel="stylesheet" type="text/css" href="http://192.168.0.59/sistemateste/css/buttons.css" />
    <link rel="stylesheet" type="text/css" href="http://192.168.0.59/sistemateste/css/table.css" />
```

Fonte: Autor (2020).

Na Figura 18 podemos observar os requisitos do ataque gerados pelo *Skipfish*, sobre um ataque do método *GET* na página HTTP do servidor, com a tentativa bruta de realizar uma tentativa de acesso ao sistema, passando uma identificação falsa no cabeçalho da página como sessão de *login*. O esperado seria o sistema tratar a tentativa com algum erro de retorno, sem ao menos contactar o banco de dados, mas o resultado foi o pior possível, a resposta dada pelo servidor *Apache* foi de que a requisição foi aceita e a conexão foi realizada, fazendo a consulta ao banco de dados para verificar o *login*. Destes resultados é possível iniciar uma sessão ao sistema sem requisição de usuário e senha, apenas forçando uma sessão no cabeçalho da página em método *GET*.

Durante os testes realizados pelo *Skipfish*, em determinado momento foi notada a lentidão da aplicação *web*, em alguns momentos foi obtido a falha de conexão com a aplicação, devido os testes de força bruta. Mesmo que os resultados de criticidade alta não obtiveram resultados, o servidor *Apache* não conseguiu controlar a demanda de requisições, apresentando algumas falhas mostradas na Figura 19.

Figura 19 – Falha do serviço do *Apache* durante o *Pentest* sem *WAF*



Fonte: Autor (2020).

Na figura 19, foi realizada a tentativa de *login* manualmente ao sistema, entrando com usuário e senha corretos para *login*, mas os mesmos deram resultados como dados incorretos, onde o *Apache* retornou um erro encontrado no serviço PHP da aplicação, retornando senha incorreta na página *login.php* em sua linha 159, mesmo rescindindo a senha novamente, o resultado de dados incorretos persistiu até que o *Skipfish* terminasse os ataques.

5.2 *Pentest OWASP ZAP* sem *WAF*

O segundo *software* de ataques executado retornou vulnerabilidades ainda maiores e com criticidades altas. O *software* aqui utilizado foi o *OWASP ZAP*, sua utilização foi dada com todas as bibliotecas contidas na configuração ativa, realizando todos os testes de vulnerabilidades possíveis pela aplicação. Nesta aplicação, utilizamos a mesma bateria com 10 testes para verificar a procedência dos resultados, a fim de verificar se haveria alguma diferença nos resultados, as quais não foram apresentadas, mantendo os resultados iguais durante a bateria de testes.

A ferramenta *ZAP* é de fácil utilização, apresentando interface gráfica e não necessitando parâmetros específicos para ataques. Na Figura 20, temos a janela da aplicação com os resultados de ataque já finalizados. Apesar dos resultados encontrados nesta ferramenta serem de maior criticidade e mais preocupantes para a empresa AWTI Sistemas de Informação, os testes realizados pelo *Pentest ZAP* foram realizados em um tempo médio com cerca de uma hora para todos os testes da bateria realizada.

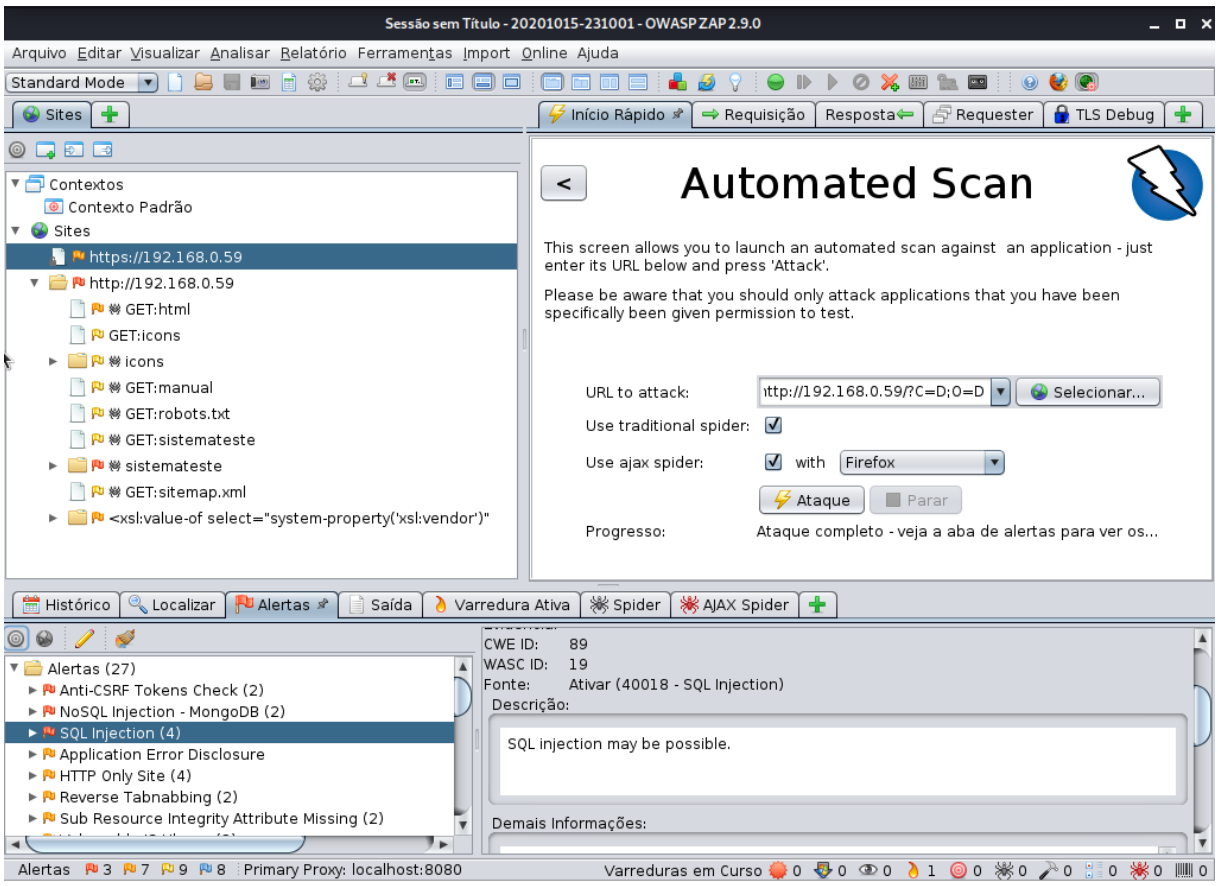
Podemos observar na Figura 20, ao centro da imagem, o campo para endereço do ataque pela aplicação de *Pentest ZAP*, logo abaixo, a possível escolha de um aplicativo de navegação HTTP para realizar os ataques por cabeçalhos, utilizando a linguagem *Ajax* como complemento para realizar os ataques e juntamente um botão para iniciar os testes de vulnerabilidades.

Os resultados do *Pentest ZAP* aparecem visualmente na aplicação, podendo ser estudados individualmente na própria aplicação, assim como podem ser exportados em página HTML como no *Skipfish,* e podendo ser exportados para um

arquivo detalhado em PDF.

A Figura 20 ainda demonstra, na barra de amostra de resultados, no inferior da aplicação, os resultados com as respectivas criticidades de vulnerabilidades, com o resultado 3 para criticidade alta, 7 de criticidade média, 9 de criticidade baixa e 8 alertas.

Figura 20 – *Pentest OWASP ZAP* sem *WAF*

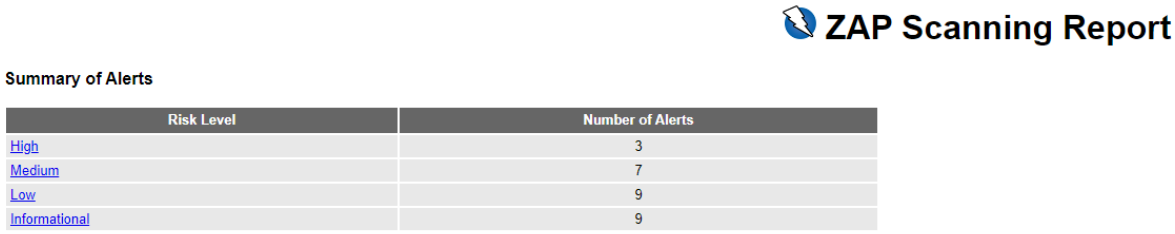


Fonte: Autor (2020).

Ao contrário da aplicação *Skipfish*, os resultados gerados pelo *Pentest* da aplicação *OWASP ZAP* contiveram criticidade alta, por sua vez, preocupantes. Dentre eles, observamos alguns resultados como *SQL injection* possíveis, estes tratados pelo instituto *OWASP* como uma das vulnerabilidades mais preocupantes.

Após exportar os resultados para uma página HTML, podemos observar melhor os resultados demonstrados conforme a Figura 21.

Figura 21 – Resultados *Pentest OWASP ZAP* sem *WAF*



ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	3
Medium	7
Low	9
Informational	9

Fonte: Autor (2020).

Na Figura 21 temos a página inicial dos resultados da aplicação *ZAP*, esta que apresenta um breve sumário, com indicações das criticidades de vulnerabilidades e os respectivos resultados de quantidade de cada criticidade. Analisando os resultados de criticidade alta, onde o primeiro resultado pode ser observado na Figura 22.

Figura 22 – Resultados *Pentest OWASP ZAP* sem *WAF mongoDB*

High (High)	**NoSQL Injection - MongoDB**
Description	MongoDB query injection may be possible.
URL	http://192.168.0.59/sistemateste//img
Method	GET
Parameter	css
Attack	css[$ne]
URL	http://192.168.0.59/sistemateste//js
Method	GET
Parameter	plupload-3.0-beta1
Attack	plupload-3.0-beta1[$ne]
Instances	2

Fonte: Autor (2020).

Apesar da aplicação de teste utilizar o banco de dados *PostgreSQL*, o *Pentest* realizado pelo *ZAP* gerou resultados com possíveis inserções de comandos de banco de dados no *MongoDB*, este é um banco de dados o qual consta na máquina virtual, mas não consta nas informações da aplicação. Todo resultado do *ZAP* é dado pela descrição do ataque, o endereço da página que gerou a vulnerabilidade, o método utilizado para realizar o ataque, a utilização dos parâmetros, a quantidade de instâncias realizadas e demais informações. Dentre as vulnerabilidades apresentadas temos também a criticidade alta de XSS, conforme Figura 23.

Figura 23 – Resultados *Pentest OWASP ZAP* sem *WAF* XSS

High (Medium)	Anti-CSRF Tokens Check
Description	A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their kn predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a us necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, cor CSRF attacks are effective in a number of situations, including: * The victim has an active session on the target site. * The victim is authenticated via HTTP auth on the target site. * The victim is on the same local network as the target site. CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have bee increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operat
URL	http://192.168.0.59/sistemateste/login
Method	POST
Evidence	<form action="http://192.168.0.59/sistemateste/login" method="post" accept-charset="utf-8" name="form_admin" id="form_admin">
URL	http://192.168.0.59/sistemateste/login
Method	GET
Evidence	<form action="http://192.168.0.59/sistemateste/login" method="post" accept-charset="utf-8" name="form_admin" id="form_admin">
Instances	2

Fonte: Autor (2020).

A vulnerabilidade apresentada na Figura 23 consta na lista do instituto *OWASP* como também umas das vulnerabilidades mais utilizadas por atacantes em sistemas *web*. Vulnerabilidade essa muito preocupante, conforme descrito nos capítulos anteriores. Observando este resultado, podemos notar uma breve descrição da vulnerabilidade, com um método diferente dos ataques anteriores, este ataque é possível ser realizado por método *POST* e *GET*, o que aumenta a vulnerabilidade. Ainda no resultado, o *software ZAP* apresenta o código da aplicação que deixou o ataque ser realizado. O último ataque com criticidade alta é apresentado na Figura 24.

Figura 24 – Resultados *Pentest OWASP ZAP* sem *WAF SQL Injection*

High (Medium)	SQL Injection
Description	SQL injection may be possible.
URL	http://192.168.0.59/sistemateste/login
Method	POST
Parameter	fl_admin
Attack	t' AND '1'='1' --
URL	http://192.168.0.59/sistemateste/login
Method	POST
Parameter	enviar
Attack	Acessar' AND '1'='1' --
URL	http://192.168.0.59/sistemateste/login
Method	POST
Parameter	senha
Attack	ZAP AND 1=1 --
URL	http://192.168.0.59/sistemateste/login
Method	POST
Parameter	senha
Attack	ZAP" AND "1"="1
Instances	4

Fonte: Autor (2020).

Da mesma maneira que a primeira vulnerabilidade de criticidade alta, nesta observamos o risco alto de inserção de comandos direto no banco *SQL*, onde a vulnerabilidade obteve 4 maneiras de *SQL injection* via endereço HTTP, apresentando todos os ataques via método *POST*, onde consta diversas maneiras de inserir comandos *SQL* com aspas simples ou duplas, apresentando resultados de que o sistema teste da empresa não provém de nenhum tratamento contra este tipo de vulnerabilidade.

Os demais resultados apresentados pela aplicação *Pentest OWASP ZAP* são de criticidade média e baixa. Riscos esses que são iguais ao *Skipfish*, apresentando apenas consultas externas da aplicação via códigos *JavaScript*. Porém, diferentemente da aplicação *Skipfish* e seus resultados, o *software ZAP* apresenta correções de códigos diretas, onde muitos dos erros de busca de valores externos podem ser corrigidos utilizando segurança SSL nas requisições HTTP, onde a aplicação passaria a ser utilizada via HTTPS, conforme consta na Figura 25.

A Figura 25 apresenta resultados de requisição sem SSL, podendo ser atacada via método *GET*, por 4 métodos diferentes. Contudo, a aplicação *ZAP* apresenta a simples correção de utilizar requisições HTTPS.

Figura 25 – Resultados *Pentest OWASP ZAP* sem *WAF* HTTPS

Medium (Medium)	HTTP Only Site
Description	The site is only served under HTTP and not HTTPS.
URL	http://192.168.0.59
Method	GET
URL	http://192.168.0.59/?C=N;O=D
Method	GET
URL	http://192.168.0.59/?C=M;O=D
Method	GET
URL	http://192.168.0.59/?C=S;O=D
Method	GET
Instances	4
Solution	Configure your web or application server to use SSL (https).
Other information	Failed to connect. ZAP attempted to connect via: https://192.168.0.59:443
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html https://letsencrypt.org/
CWE Id	311
WASC Id	4
Source ID	1

Fonte: Autor (2020).

Ao contrário dos testes realizados no *Skipfish*, aos quais apresentaram lentidão no servidor *Apache*, apresentando até mesmo alguns erros de aplicação, os testes

realizados pelo *Pentest OWASP ZAP* não apresentaram instabilidades de performance ou uso da aplicação.

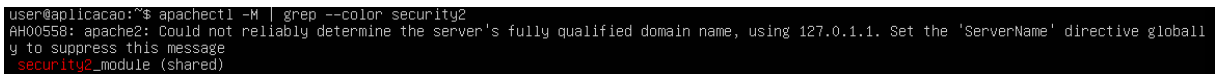
Com estes resultados apresentados pelos dois *Pentest* temos alguns resultados de impactos com criticidade alta, vulnerabilidades que são prejudiciais para a aplicação e para os clientes com dados cadastrados, visando leis brasileiras como a LGPD a qual entrou em vigor recentemente. Com estes resultados preocupantes nos testes sem um *WAF* para proteger a aplicação *web*, observamos a real necessidade de aplicar um *WAF* e proteger a aplicação, com isso instalamos e configuramos o *ModSecurity* e as regras CRS de *blacklist* no servidor *Apache*.

5.3 Instalação *ModSecurity*

Para aplicar a segurança necessária para evitar os resultados anteriores, utilizamos o módulo *ModSecurity*, este que, descrito em capítulos anteriores, se trata de um *WAF* do *Apache*.

Sua instalação é implementada pelo gerenciador de pacote do *Linux* APT-*GET*. Acessando o terminal da máquina virtual, digitamos o comando *sudo apt install libapache2-mod-security2*. Desta forma, é instalado o módulo *ModSecurity* e suas dependências. Para termos a certeza de que o módulo foi instalado, executamos o comando *apachectl -M* | *grep –color security2,* com este comando podemos observar o resultado *shared* (compartilhado), conforme a Figura 26, obtendo a certeza de que o módulo está instalado e compartilhado.

Figura 26 – *ModSecurity*

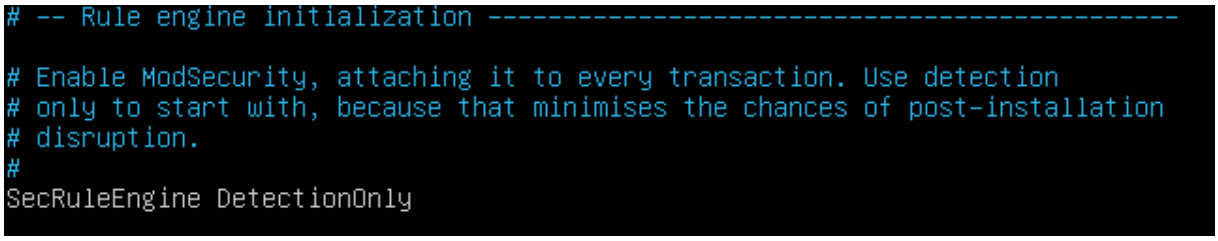


Fonte: Autor (2020).

O próximo passo da instalação do *ModSecurity* foi ativar o modo de verificação de vulnerabilidades, que, por sua vez, vem ativo apenas para detectar os pacotes, sem intervir. Para isso, no Terminal do *Linux* abrimos o arquivo */etc/modsecurity/modsecurity.conf-example*, modificamos a linha de comando *SecRuleEngine DetectionOnly* para *SecRuleEngine On*. Após, salvamos o arquivo

com nome *modsecurity.conf*, desta forma, o *ModSecurity* passa a intervir nos pacotes de dados, caso detecte como vulnerabilidade, conforme Figura 27 (KUMAR, 2020).

Figura 27 – *ModSecurity* regra ativado



Fonte: Autor (2020).

Após este processo, foi necessário realizar a reinicialização do *Apache* para que ele recarregasse o módulo *ModSecurity* e suas configurações, com o comando *sudo /etc/init.d/apache2 restart*.

Com estes comandos tivemos o *ModSecurity* em execução, porém sem as regras da *blacklist* para atuar e detectar as vulnerabilidades, para isso, o próximo passo foi adquirir e implementar as regras *OWASP ModSecurity Core Rule Set* (CRS). Para isso, teve que se obter as regras da *internet* via comando de terminal, regras encontradas no *site* da *OWASP*, como mencionado em capítulos anteriores. Para isso, criamos uma pasta temporária para baixar e descompactar o arquivo, para adquirir o arquivo, efetuamos o comando *sudo git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git*, conforme a Figura 28.

Figura 28 – Baixando regras CRS

Fonte: Autor (2020).

Após adquirir os arquivos, foi copiado o arquivo de configuração para o *ModSecurity*, e, assim, foram habilitadas as regras CRS junto ao módulo *WAF*, executando o comando *sudo cp /usr/share/owasp-modsecurity-crs/ /usr/share/modsecurity-crs/*. Após os arquivos serem copiados para a respectiva pasta do *ModSecurity*, o arquivo de segurança do *Apache* foi editado para incluir as regras CRS da *blacklist OWASP*, executando o comando *sudo nano /etc/apache2/mods-*

enabled/security2.conf. Neste arquivo, foram inseridas duas linhas de comando, o caminho do arquivo de configuração e após, o caminho das regras de *blacklist*. Por fim, foi reiniciado novamente o servidor *Apache*, já com o *ModSecurity* configurado e executando as regras CRS *OWASP* (KUMAR, 2020).

A fim de realizar segurança, foi implementado também o módulo adicional do *Apache Mod Evasive*, o qual funciona em conjunto com *ModSecurity*. Sua função é fazer a leitura dos registros do *ModSecurity*, em casos de vulnerabilidade com alto risco, o módulo efetua o bloqueio do remetente/atacante pelo tempo determinado nas configurações, se ativada a função de relatar, a função envia o registro do *ModSecurity*, junto com a ação tomada, para o *e-mail* definido. Para essa implementação, foi executada a instalação também pelo gerenciado de pacotes do *Linux,* o comando *sudo apt-get install libapache2-mod-evasive.*

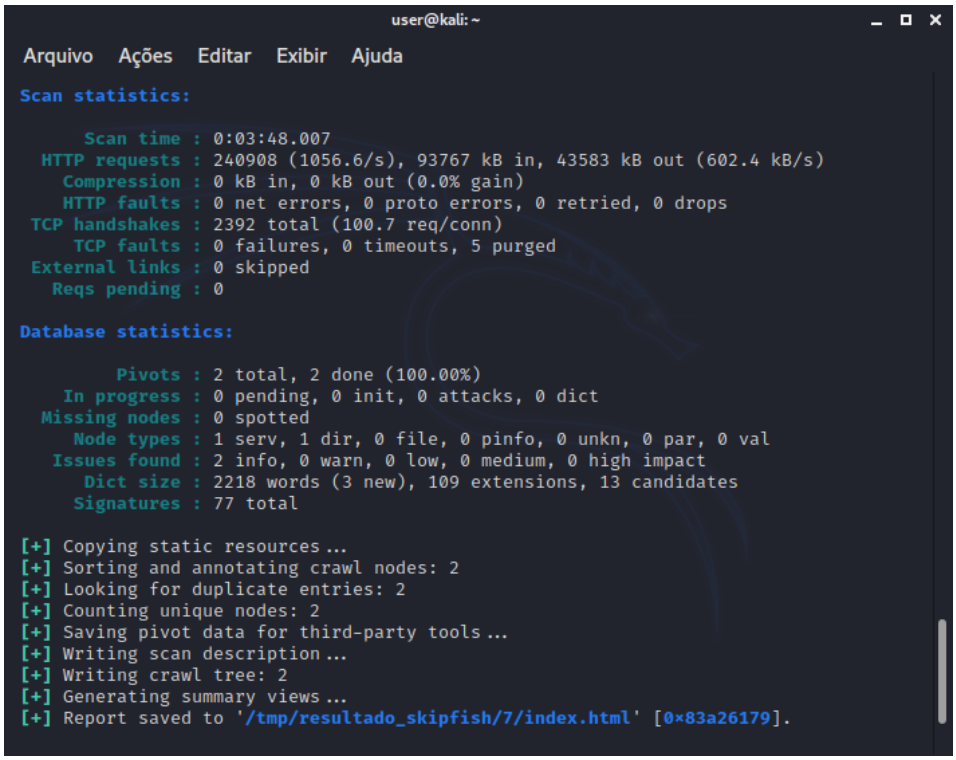
Para ativar de fato o módulo no *Apache*, foi efetuado o comando *sudo a2enmod evasive.* Após, foi configurado o arquivo */etc/apache2/mods-available/mod-evasive.conf*, com os parâmetros necessários para o módulo efetuar a devida proteção, e, por fim, foi reiniciado novamente o *Apache*, desta forma, foi obtida a segurança com o *ModSecurity* e *Mod Evasive* configurados (STAFF, 2020).

Para ter certeza, foram rodadas novamente as baterias de teste do S*kipfish* e *OWASP ZAP*, agora com WAF implementado.

5.4 *Pentest Skipfish* com *WAF*

Os testes executados após a instalação e implementação do *ModSecurity* e *Mod Evasive* foram executados, mantendo o padrão de ataque, com uma bateria de 10 testes com os mesmos parâmetros dos testes sem o *WAF* implementado. Os resultados iniciais já foram satisfatórios, conforme a Figura 29, obtendo um total de zero ataques de criticidade alta, média e baixa possíveis. Foi percebido também o tempo decorrido de aproximadamente 4 minutos em cada teste, devido ao bloqueio de IP do atacante pelo tempo determinado na configuração do *Mod Evasive*.

Figura 29 – *Skipfish* com *WAF*



Fonte: Autor (2020).

Na Figura 29, assim como nos outros testes da bateria realizada, todos tiveram os mesmos resultados e terminaram com menos de 5 minutos de ataque, sendo encerrados devido a não ter mais comunicação com o servidor *Apache*. Na Figura 30, temos os registros de acessos ao *Apache*, onde os ataques com IP 192.168.0.220, respectivamente, o *Kali Linux* com *Skipfish*, fez requisições em método *GET* pelo navegador *Mozilla*, mas obteve um retorno de página com erro HTML 403 (Falha de arquivo ou permissão de acesso) do servidor 192.168.0.164, devido ao bloqueio dos pacotes no *ModSecurity*.

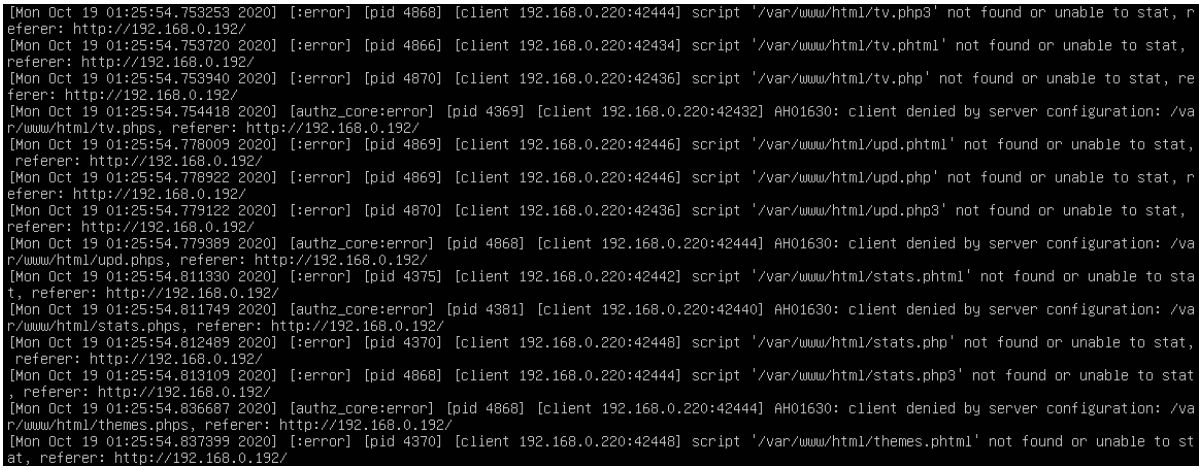
Figura 30 – Registro de acesso do *Apache*

Fonte: Autor (2020).

No mesmo teste, ainda observamos que nos registros de erros do *Apache*, consta os mesmos ataques do *Skipfish*, conforme Figura 31, acessos bloqueados pelo

Mod Evasive, bloqueando o acesso do cliente 192.168.0.220 (*Kali Linux*).

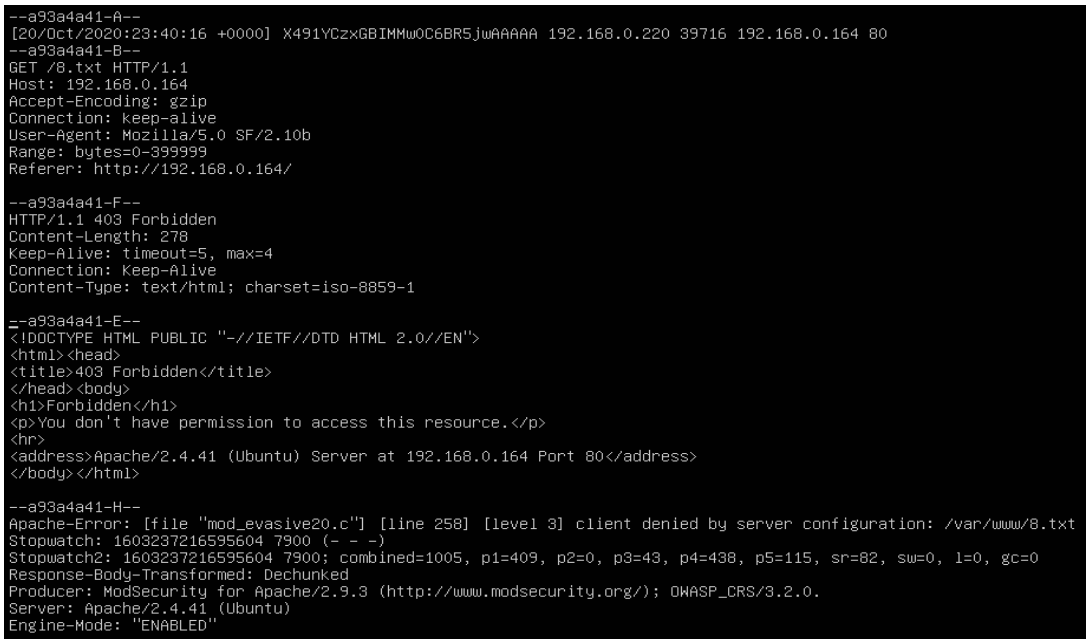
Figura 31 – Registro de erros do *Apache*



Fonte: Autor (2020).

Durante os mesmos testes, também foram acompanhados os registros do *ModSecurity,* conforme Figura 32, o qual apresentou os resultados de bloqueio do cliente 192.168.0.220 (*Kali Linux*), após uma requisição do método *GET* ter caído em uma das regras com nome *modevasive20*. Pode-se notar, no registro do *ModSecurity*, a requisição para o IP 192.168.0.64, na porta 80, pelo navegador *Mozilla*/5.0, via método *GET* para o arquivo /8.txt, onde obteve um retorno com erro HTML com a página de erro 403 *Forbiden,* apresentada pelo servidor *Apache* do IP 192.168.0.164, na porta 80. Abaixo, ainda temos os parâmetros do ataque, com a regra na qual o ataque foi barrado, a linha 258 representando a classe da regra, novamente o arquivo o qual queria acessar, as regras com identificação para *ModSecurity*, o registro da *blacklist* implementado, e, por fim, o método de filtro *WAF* com parâmetro ativo.

Figura 32 – Registro de erros do *Apache*



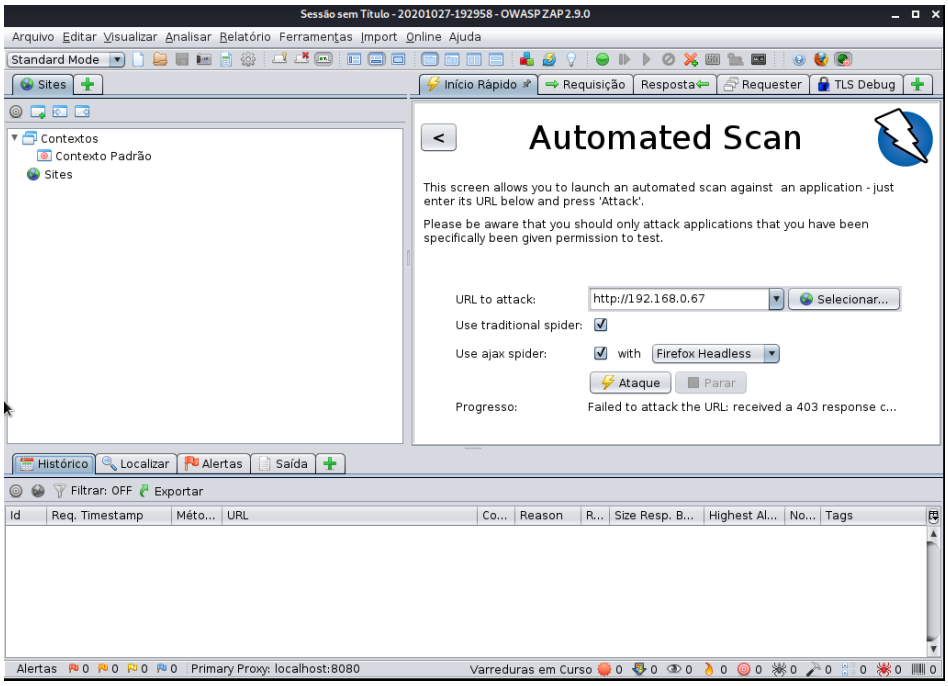
Fonte: Autor (2020).

Seguindo a bateria de *Pentest*, após ter os resultados obtidos pelo *Skipfish*, foi efetuada a bateria de testes com *OWASP ZAP*, para assim efetivar os resultados e terminar as conclusões, onde foram comparados os resultados obtidos, efetivando a segurança que foi ganhada com a implementação de um *WAF* no *Apache*.

5.5 *Pentest OWASP ZAP* com *WAF*

Com a aplicação do *WAF* implementada, a bateria de testes pelo *OWASP ZAP* se deu de forma nula. Logo ao iniciar cada teste da bateria, com menos de 10 segundos em todas as tentativas da bateria de testes, o *Pentest* foi bloqueado e se encerrou com erro 403 como página de retorno, erro que se deve a não encontrar o servidor *web*, conforme a Figura 33.

Figura 33 – *OWASP ZAP* com *WAF*



Fonte: Autor (2020).

Conforme na Figura 33, pode-se observar que os testes da bateria com *WAF* implementado e utilizando o *OWASP ZAP* para atacar a aplicação da empresa AWTI Sistemas de Informação, os resultados foram os melhores possíveis, onde o atacante não conseguiu nenhuma vulnerabilidade e foi bloqueado, impedindo que os ataques continuassem e deixassem o sistema lento ou instável.

Nos registros do *ModSecurity* podemos observar o bloqueio acontecendo, conforme a Figura 34.

Figura 34 – Registro *ModSecurity* com *WAF*

Fonte: Autor (2020).

Conforme pode-se observar na Figura 34, à qual se refere em um registro do *ModSecurity*, os ataques de *Pentest* registrados como vulnerabilidades de severidades perigosas e críticas pelo ModSecurity, são regras definidas pela *blacklist* CRS que ocorreram durante o *Pentest*, até o momento em que o *Mod Evasive* bloqueia a conexão do atacante.

5.6 Comparando os resultados

Com a finalidade de observar os resultados, os mesmos foram comparados em quadros, para concluir e demonstrar à empresa AWTI Sistemas de Informação os estudos da aplicação do *WAF* e sua eficiência, junto dos resultados, de uma forma abrangente e estatística.

Utilizando a análise estatística descritiva de dados, foram comparados os resultados das vulnerabilidades, em nível de criticidade de cada resultado

apresentado nas baterias de *Pentest*, conforme o Quadro 2 (EJE, 2019).

Quadro 2 – Resultado de criticidade

Criticidade	Alta	Média	Baixa	Informações
Skipfish* sem *WAF	0	2	4	40
ZAP* sem *WAF	3	7	9	9
Skipfish* com *WAF	0	0	0	2
ZAP* com *WAF	0	0	0	0

Fonte: Autor (2020).

No Quadro 2, observa-se que mesmo em um sistema de cadastro simples, oferecido para testes pela empresa AWTI Sistemas e Informação, foi possível ter criticidades médias e baixas em um simples teste de ataque com a ferramenta *Skipfish*, e resultados com criticidade ainda maiores nos testes de vulnerabilidades pela ferramenta *OWASP ZAP*, resultando 3 vulnerabilidades de criticidade alta, 7 de criticidade média e 9 de criticidade baixa.

Ainda no Quadro 2, nota-se que, após o uso do *WAF ModSecurity* em conjunto com *Mod Evasive*, houve uma queda das vulnerabilidades, zerando os resultados, neste caso de estudo, protegendo 100% a situação de laboratório diante das duas ferramentas de *Pentest*.

Com os atuais resultados do Quadro 2, foi notado que a ferramenta de *Pentest* *OWASP ZAP* se saiu melhor, com uma performance melhor nos testes realizados, demandando menos tempo de execução, com resultados que podem ser exportados para um arquivo PDF, com relatório completo de cada vulnerabilidade, demonstrando ao cliente, e, por fim, notado principalmente as criticidades encontradas no mesmo sistema, abrangendo uma avaliação melhor das vulnerabilidades das aplicações *web*.

5.7 Resultados de testes não satisfatórios

Durante os testes de implementação, houve algumas dificuldades na implementação do *Apache* e acessos externos ao banco de dados *PostgreSQL*. Resultando em criticidades graves, conforme o Quadro 3, as quais são tratadas nos servidores da empresa AWTI Sistema de Informação, devido a essas configurações

de acessos e permissões serem tratadas nos servidores da empresa, os resultados foram anotados para uma comparação de dados, mas descartados para resultados a serem apresentados nas correções para a empresa.

Quadro 3 – Resultados do *Apache* e *PostgreSQL* sem configurações

Criticidade	Alta	Média	Baixa	Informações
Skipfish	6	4	2	44

Fonte: Autor (2020).

Seguindo as sugestões de implementação de segurança ao *Apache*, conforme mencionado anteriormente, foi implementado o *ModSecurity*, com as regras CRS, na última versão disponível, mas houve problemas quanto ao acesso à aplicação não disponibilizando a navegação da aplicação. Nos registros do *Apache*, a requisição de acesso não obtinha registro algum, mas nos registros do *ModSecurity*, foram encontrados registros com bloqueios e exigências da utilização de SSL para HTTP, forçando a implementação e navegação via HTTPS, caso contrário, as regras de *blacklist* reconheciam como alta criticidade, bloqueando o acesso HTTP ao *Apache*.

Para implementar regras CRS, sem que fosse necessário a modificação da lista de regras, e ainda mantivesse uma boa segurança à aplicação *web*, foram trocadas as regras CRS para uma versão anterior, versão 2.18. Voltando o acesso à aplicação, sem ter interrupções nas funções da aplicação. Mas não foi obtido um bom resultado nos testes de vulnerabilidades, apresentando algumas possíveis criticidades médias e baixas em ambas as ferramentas de *Pentest*, inclusive com resultados diferentes, no mesmo teste executado pelo *Skipfish*, conforme Quadro 4.

Quadro 4 – Resultados *ModSecurity* e CRS 2.18

Criticidade	Alta	Média	Baixa	Informações
***Skipfish Pentest* 1**	0	1	1	30
***Skipfish Pentest* 2**	0	2	3	36
OWASP ZAP	0	6	8	8

Fonte: Autor (2020).

Sem o efetivo resultado esperado, passou-se a utilizar uma versão intermediária, disponível no *site* da *Core Rule Set*, implementando a versão 3.2 das regras CRS, executando e validando os resultados anteriores apresentados, com um resultado satisfatório e uma funcionalidade sem modificações na aplicação *web*.

6 CONSIDERAÇÕES FINAIS

O presente trabalho abordou a implementação do *ModSecurity* como módulo *WAF* do *Apache*, com regras CRS para detectar as vulnerabilidades e minimizar os riscos de segurança nas aplicações *web* da empresa AWTI Sistemas de Informação. Após o término da implementação do *ModSecurity* e *Mod Evasive*, com os testes de ataque realizados e documentados, a real constatação do funcionamento da solução, diante das leis de seguranças Brasileiras, foi de uma proposta de configuração com um resultado obtido nas comparações muito satisfatórios, zerando as vulnerabilidades de ataque.

O *WAF ModSecurity* se mostrou muito satisfatório e robusto quanto aos testes de ataques realizados, atendendo os requisitos e protegendo as aplicações *web* do servidor Apache nas portas 80 e 443, mesmo para uma aplicação *web* que não possui nenhum tratamento de segurança, mas necessita uma atenção para a versão das regras CRS aplicadas, essas que, se muito antigas, não provêm de todas as seguranças necessárias, podendo deixar algumas vulnerabilidades passarem. E, se as regras forem muito atuais, é necessário que a aplicação *web* e o servidor *Apache* atendam uma configuração de segurança para que não haja bloqueios do *ModSecurity,* por detectar como vulnerável a navegação entre servidor e cliente, e bloquear o acesso do cliente, mesmo que este não seja um atacante.

Com uma versão específica das regras CRS, foi possível manter a navegação do cliente com a aplicação *web*, sem ter restrições de navegação e ao mesmo tempo garantir a segurança da mesma contra os ataques. Além da segurança contra os ataques, nos testes realizados percebeu-se que o *ModSecurity* também proporciona uma segurança de performance contra a lentidão ou possíveis instabilidades de

acesso, caso o sistema seja atacado.

As regras CRS da *OWASP* e *ModSecurity* obtiveram resultados positivos, zerando os resultados de criticidade alta, média e baixa. Não houve a necessidade de procurar outras regras, apenas uma adequação da versão das regras CRS, para proporcionar uma boa estabilidade e uso da aplicação *web* existente, sem que fosse necessário modificações nas regras CRS ou na aplicação *web*.

Com essa configuração de *WAF* e regras de CRS, foi obtido o conhecimento de que um servidor *Apache* com *WAF* pode funcionar para uma aplicação, mas pode ser problema para outra. Desta forma, o uso deste *WAF* implementado não atingiu um resultado satisfatório. Para utilizá-lo em outras aplicações pode ser necessário ajustes, conforme a necessidade do cliente, desativando algumas regras de *blacklist*, atendendo os requisitos de cada sistema, desta forma podendo ser utilizado em demais clientes.

Por fim, o estudo da ferramenta *WAF* proporcionou uma ótima segurança contra ataques em aplicações *web nas portas 80 e 443, HTTP e HTTPS respectivamente*, mas a necessidade da aplicação também tratar em seu código fonte as vulnerabilidades com exigências, como o uso de SSL em HTTP, senhas de acesso seguras, tratamento de ataques nos acessos a cada banco de dados e outras vulnerabilidades, é necessária para uma boa implementação do *WAF*, mantendo-o sempre atualizado, para defender a todas as vulnerabilidades atuais. Caso contrário, o *WAF* não poderá ser atualizado, pois pode enfrentar problemas com a aplicação *web*, bloqueando recursos da aplicação ou até mesmo todo acesso a mesma, sendo necessário modificações nas regras de *blacklist* que bloquearam a conexão para atender a baixa segurança da aplicação, abrindo brechas de segurança e podendo ter vulnerabilidades na aplicação.

A ferramenta *ModSecurity* implementada é gratuita e atende os requisitos das mais diversas vulnerabilidades, porém uma implementação em uma aplicação maior ou ainda com mais aplicações implementadas no mesmo servidor, diante de um cenário com *ModSecurity*, avaliando o desempenho e segurança, com ferramentas avaliativas para tal finalidade, diante deste cenário, é um estudo futuro para encontrar uma solução pelos mais diversos servidores de pequenas e médias empresas que possuem mais de uma aplicação no mesmo servidor.

REFERÊNCIAS

ASSUNÇÃO, Marcos Flávio Araújo. **Wireless hacking**: ataque e segurança de redes sem fio wi-fi. São Paulo: Visual Books, 2013.

AUGUSTO, Cassio. Linux cresceu 50% no último ano e ultrapassou 3,37% do mercado. **Ninja do Linux**, 2017. Disponível em: <http://ninjadolinux.com.br/linux-cresceu-50-por-cento-no-ultimo-ano/>. Acesso em: 23 mai. 2020.

CANONICAL, Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd. **Tutorial - Install and Configure Apache**. Disponível em: <https://ubuntu.com/tutorials/install-and-configure-apache#2-installing-apache> Acesso em: 18 ago. 2020.

COSTA, José Victor Pereira. Análise de Vulnerabilidades de Segurança em Portais de Governos Eletrônicos. **Universidade Federal de Uberlândia**, Uberlândia, 2017. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/20400/6/AnaliseVulnerabilidadesSeguranc%cc%a7a.pdf>. Acesso em: 01 jun. 2020.

CERT.br. **Estatísticas dos incidentes reportados ao CERT.br**, 2019. Disponível em: <https://www.cert.br/stats/incidentes/>. Acesso em: 29 mai. 2020.

EJE. **Análise Estatística Descritiva**, 2019. Disponível em: <https://ejeconsultoria.com.br/2019/08/16/analise-estatistica-descritiva/>. Acesso em: 27 out. 2020.

EVEO. **WAF** – O que é e para que serve o Web Application Firewall?, 2018. Disponível em: <https://www.eveo.com.br/blog/waf-web-application-firewall>. Acesso em: 23 mai. 2020.

FOLINI, Christian. **Including OWASP ModSecurity Core Rule Set**. Netnea, 2019. Disponível em: <https://www.netnea.com/cms/apache-tutorial-7_including-modsecurity-core-rules.>. Acesso em: 03 jun. 2020.

FOLINI, Christian; RISTIC, Ivan. **ModSecurity Handbook**: getting started. 2. ed. 2018. Disponível em: <https://www.feistyduck.com/library/modsecurity-handbook-2ed-free/online>. Acesso em: 03 jun. 2020.

FORD, Jerry Lee. **Manual completo de firewalls pessoais**: Tudo o que você precisa saber para proteger o seu computador. São Paulo: Makron Books, 2002.

FRANZINI, Fernando. **Vulnerabilidades de aplicativos web**. iMasters, 2009. Disponível em: <https://pt.scribd.com/document/100235424/iMasters-Vulnerabilidades-de-Aplicativos-Web>. Acesso em: 01 jun. 2020.

GALVÃO, Michele da Costa. **Fundamentos em segurança da informação**. São Paulo: Editora Pearson, 2015.

GOCACHE. **Melhores práticas de segurança em aplicações web**, 2017. Disponível em: <https://www.gocache.com.br/seguranca/seguranca-em-aplicacoes-web>. Acesso em: 29 mai. 2020.

HAINZENREDER, Uilian Mengue. **WSS-SOC**: um conjunto de diretrizes para estruturação de centro de operações de segurança. Universidade Feevale, Novo Hamburgo, 2019. Disponível em: <https://tconline.feevale.br/NOVO/tc/files/0001_4684.pdf>. Acesso em: 29 mai. 2020.

KUMAR, Pankaj. Curious Viral. **How To Install and Configure ModSecurity On Ubuntu 20.04.** Disponível em: <https://curiousviral.com/install-and-configure-mod-security/>. Acesso em: 15 set. 2020.

KUROSE, Jim; ROSS, Keith. **Redes de computadores e a internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson, 2014.

LIMA, Telma Cristiane Sasso; MIOTO, Regina Célia Tamaso. **Procedimentos metodológicos na construção do conhecimento científico**: a pesquisa bibliográfica. Rev Katál, Florianópolis, v. 10, p. 37-45, 2007. Disponível em: <https://www.scielo.br/pdf/rk/v10nspe/a0410spe>. Acesso em: 09 jun. 2020.

LOUIS, Jerry. **Detection of Session Hijacking**. 2011. University of Bedfordshire. Disponível em: <https://uobrep.openrepository.com/bitstream/handle/10547/211810/louis2011.pdf?sequence=1&isAllowed=y>. Acesso em: 27 out. 2020.

MONTEIRO, Nuno Miguel da Silva. **Estudo de vulnerabilidades em aplicações web e o seu reflexo em domínios portugueses.** Instituto Superior de Engenharia do Porto – ISEP, 2015. Disponível em: <https://core.ac.uk/download/pdf/47142754.pdf>. Acesso em: 23 mai. 2020.

MONTEVERDE, Wagner Aparecido; CAMPIOLO, Rodrigo. **Estudo e análise de vulnerabilidades web.** Soc. Bras. de Computação, Campo Mourão – PR, 2014. Disponível em: <https://pdfs.semanticscholar.org/05cc/81abcc8393a4dd16aed3ef28f0977a54da18.pdf>. Acesso em: 29 mai. 2020.

MORETTI, Isabella. **Metodologia de Pesquisa TCC: passo a passo com exemplos,** 2017. Disponível em: <https://viacarreira.com/metodologia-de-pesquisa-do-tcc/#Pesquisa_Exploratoria>. Acesso em: 19 Set. 2020.

NAKAMURA, Emilio Tissato; GEUS, Paulo Lício de. **Segurança de redes em**

ambientes cooperativos. São Paulo: Novatec Editora, 2007.

NETO, Alfredo Del Fabro. **Utilização de firewall de aplicação no processo de desenvolvimento de sistemas web**. Santa Maria: Universidade Federal de Santa Maria, 2013.

NOLETO, Caio. **Segurança da informação: o que é e quais os 5 principais pilares?** 2020. Disponível em: <https://blog.betrybe.com/tecnologia/seguranca-da-informacao/>. Acesso em: 11 dez. 2020.

OWASP. **Testing for Cross site scripting**. 2013. Disponível em: <https://www.owasp.org/index.php/Testing_for_Cross_site_scripting>. Acesso em: 29 mai. 2020.

OWASP top 10. **The ten most critical web application security risks**, 2017. Disponível em: <https://owasp.org/www-pdf-archive/OWASP_Top_10-2017-pt_pt.pdf>. Acesso em: 01 jun. 2020.

OWASP, **The Open Web Application Security Project**, versão 1.3, 2012. Disponível em: <https://owasp.org/www-pdf-archive/OWASP_SCP_v1.3_pt-PT.pdf>. Acesso em: 29 abr. 2020.

PHP.COM.BR. **Instalação do PHP em Ubuntu Linux**. Veja o passo a passo de como instalar o PHP no Ubuntu 18.04. Disponível em: <https://php.com.br/14?instalacao-do-php-em-ubuntu-linux>. Acesso em: 18 ago. 2020.

POSTGRESQL. The PostgreSQL Global Development Group. **Linux downloads (Ubuntu). PostgreSQL Apt Repository**. Disponível em: <https://www.postgresql.org/download/linux/ubuntu/>. Acesso em: 19 ago. 2020.

PRADA, Rodrigo. **O que é exploit?** Tecmundo, 2008. Disponível em: <http://www.tecmundo.com.br/seguranca/1218-o-que-e-exploit-.htm>. Acesso em: 03 jun. 2020.

PROCESSING phases of Modsecurity. **Malware Expert**, commercial modsecurity rules, 2017. Disponível em: <https://malware.expert/modsecurity/processing-phases-modsecurity>. Acesso em: 03 jun. 2020.

RISTIC, Ivan. **ModSecurity Handbook**. Feisty Duck, 2010. Disponível em: <http://index-of.es/Networking/MOD%20SECURITY%20handbook.pdf>. Acesso em: 29 abr. 2020.

ROHR, Altieres. **Entenda e proteja-se do ataque de 'credential stuffing', usado por hackers em vazamentos de dados**. 2019. Disponível em: <https://g1.globo.com/economia/tecnologia/blog/altieres-rohr/post/2019/02/06/entenda-e-proteja-se-do-ataque-de-credential-stuffing-usado-por-hackers-em-vazamentos-de-dados.ghtml>. Acesso em: 27 out. 2020.

SILVA, Igor. **Tendência de TI e Desenvolvimento Web para 2020**. Scriptcase blog, 2020. Disponível em: <https://www.scriptcaseblog.net/pt/design-ux-pt/tendencia-de-ti-e-desenvolvimento-web-para-2020/>. Acesso em: 11 dez. 2020.

SILVEIRA, Klaubert Herr da. **ModSecurity**: firewall opensource para aplicações web (WAF). OWASP, The Open Web Application Security Project, 2012. Disponível em: <https://pt.slideshare.net/owaspbsb/owasp-bsb-modsecurityklaubertherrv2012>. Acesso em: 29 mai. 2020.

SKIPFISH. **Web application security scanner for XSS, SQL injection, Shell injection**. GBHackers On Security, 2018. Disponível em: <https://gbhackers.com/skipfish-web-application-security-scanner>. Acesso em: 03 jun. 2020.

STAFF, Editorial. **How to Configure mod_evasive With Apache on Debian/Ubuntu**, 2020. Disponível em: <https://brightwhiz.com/mod_evasive-security-setup/>. Acesso em 24 set. 2020.

STALLINGS, William. **Criptografia e segurança de redes**. 6. ed. São Paulo: Pearson, 2014.

STALLINGS, William; BROWN, Lawrie. **Instructor Solutions Manual for Computer Security: Principles and Practice.** 3. Ed. Pearson, 2015.

STEIN, Jonas Eduardo. **Metodologia de configuração de vulnerabilidades para o modsecurity**. Universidade Federal de Santa Maria, Santa Maria, 2013. Disponível em: <http://www.redes.ufsm.br/docs/tccs/Jonas_Stein.pdf>. Acesso em: 09 mai. 2020.

TUMELERO, Naína. **Pesquisa aplicada**: material completo, com exemplos e características. Mettzer, 2019. Disponível em: <https://blog.mettzer.com/pesquisa-aplicada>. Acesso em: 09 jun. 2020.

WEB Application Firewall (WAF): A handbook to understand web application firewall. **IPA** (Information-Technology Promotion Agency), Japão, 2011. Disponível em: <https://www.ipa.go.jp/files/000017313.pdf>. Acesso em: 03 jun 2020.

WILHELM, Thomas. **Professional penetration testing**: creating and learning in a hacking lab. Elsevier, 2. ed., Waltham – MA, 2013.